TPM Main
Part 3 Commands

Specification Version 1.2
Level 2 Revision 94
29 March 2006

Contact: tpmwg@trustedcomputinggroup.org

TCG Published

[redacted]

Disclaimer

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Acknowledgement

TCG wishes to thank all those who contributed to this specification. This version builds on the work published in version 1.1 and those who helped on that version have helped on this version.

A special thank you goes to the members of the TPM workgroup who had early access to this version and made invaluable contributions, corrections and support.

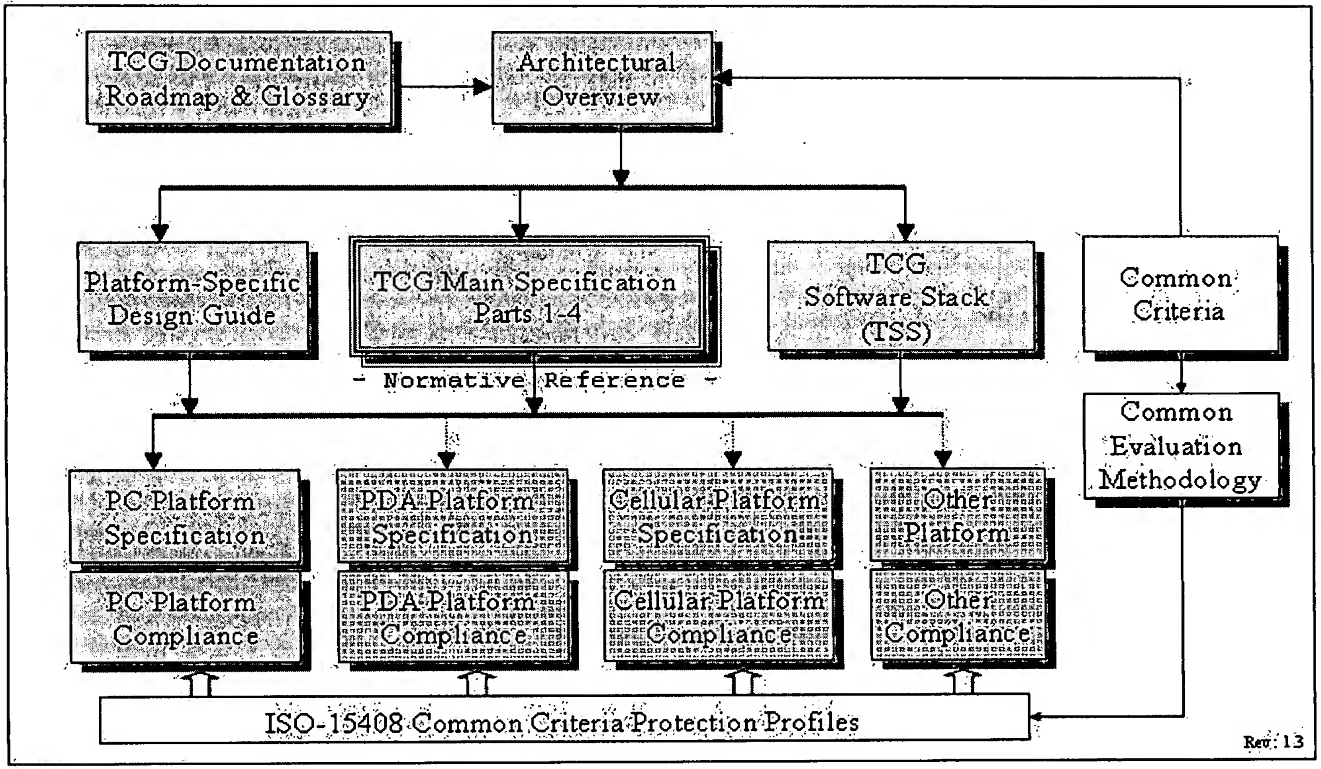
David Grawrock

TPM Workgroup chair

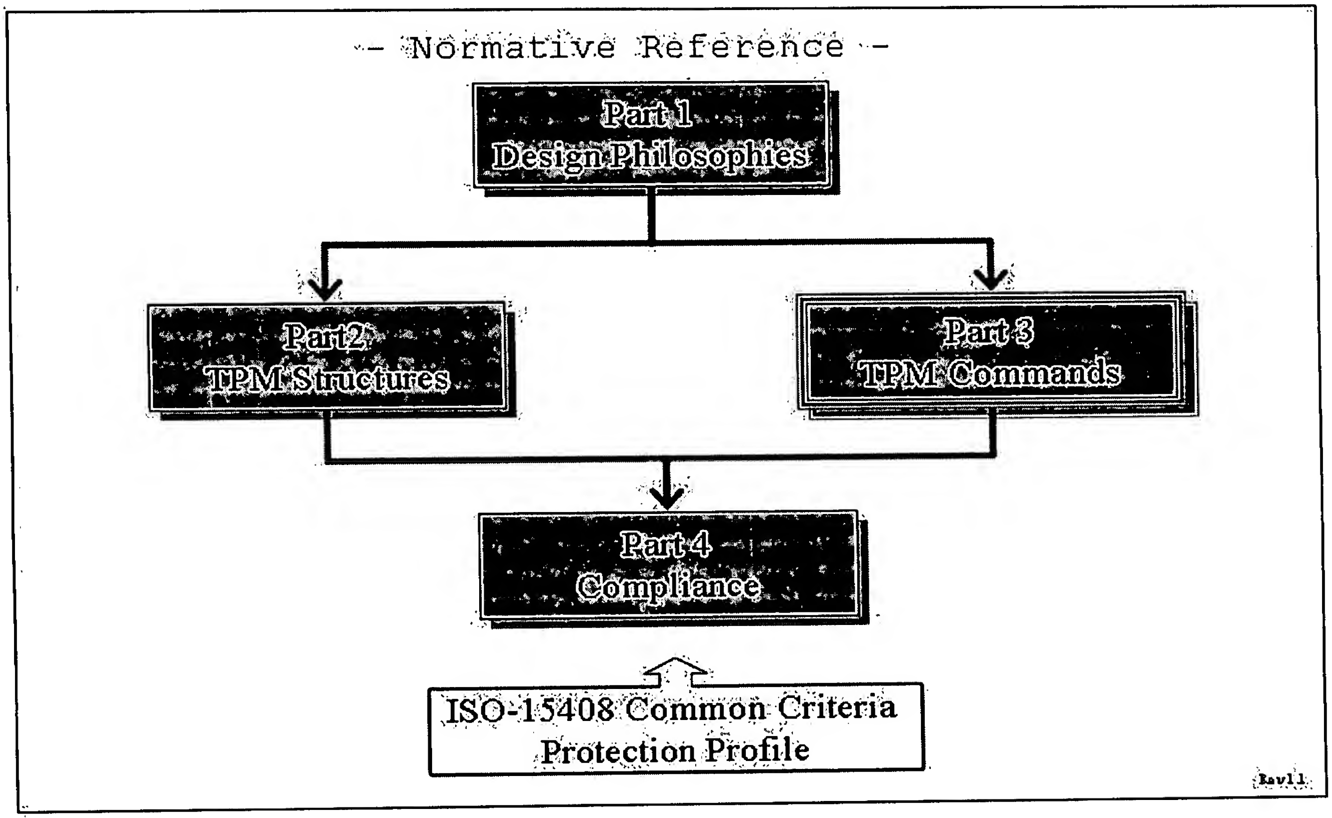
Change History

Version	Date	Description
Rev 50	Jul 2003	Started 01 Jul 2003 by David Grawrock Breakup into parts and the merge of 1.1 commands
Rev 63	Oct 2003	Change history tied to part 1 and kept in part 1 (DP)
Rev 71	Mar 2004	Change in terms from authorization data to AuthData.
Rev 91	Sept 2005	The following modifications were made by Tasneem Brutch: ▪ Update to section 6.2 informative, for TPM_OwnerClear. ▪ Addtion of action item 15, to section 6.2, for TPM_OwnerClear. ▪ Addition of "MAY" to section 20.1, TPM_NV_DefineSpace, Action 1(a). ▪ Addition of a new Action (4) to Section 20.2, TPM_NV_WriteValue ▪ Addtion of a new Action (3) to Section 20.4, TPM_NV_ReadValue. ▪ Typo corrected in Section 21.1 ▪ Moved TPM_GetCapabilityOwner from Section the Deleted Commands (section 28.1) to section 7.3. Added information on operands, command description and actions from Rev. 67.
Rev 92	Sept 2005	Section 7.3 TPM_GetCapabilityOwner Ordinal was added to the outgoing params, which is not returned but is typically included in outParamDigest.
Rev 92	Sept 2005	Corrected a copy and paste error: Part 3 20.2 TPM_NV_WriteValue Removed the Action "3. If D1 -> TPM_NV_PER_AUTHREAD is TRUE return TPM_AUTH_CONFLICT"
Rev 93	Sept. 2005	Moved TPM_CertifySelfTest command to the deleted section.

TCG Doc Roadmap – Main Spec


TCG Documentation Roadmap & Glossary
Architectural Overview
Platform-Specific Design Guide
TCG Main Specification Parts 1-4
TCG Software Stack (TSS)
Common Criteria
- Normative Reference -
Common Evaluation Methodology
PC Platform Specification
PDA Platform Specification
Cellular Platform Specification
Other Platform
PC Platform Compliance
PDA Platform Compliance
Cellular Platform Compliance
Other Compliance
ISO-15408 Common Criteria Protection Profiles
Rev: 13

TCG Main Spec Roadmap



- Normative Reference -

Part 1
Design Philosophies

Part2
TPM Structures

Part 3
TPM Commands

Part 4
Compliance

ISO-15408 Common Criteria
Protection Profile

Rev11

Table of Contents

End of Introduction do not delete

1. Scope and Audience

The TPCA main specification is an industry specification that enables trust in computing platforms in general. The main specification is broken into parts to make the role of each document clear. A version of the specification (like 1.2) requires all parts to be a complete specification.

This is Part 3 the structures that the TPM will use.

This document is an industry specification that enables trust in computing platforms in general.

1.1 Key words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in the chapters 2-10 normative statements are to be interpreted as described in [RFC-2119].

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. You will encounter two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, you can consider it of the kind normative statements.

For example:

Start of informative comment:

This is the first paragraph of 1–n paragraphs containing text of the kind informative comment ...

This is the second paragraph of text of the kind informative comment ...

This is the nth paragraph of text of the kind informative comment ...

To understand the TPM specification the user must read the specification. (This use of MUST does not require any action).

End of informative comment.

This is the first paragraph of one or more paragraphs (and/or sections) containing the text of the kind normative statements ...

To understand the TPM specification the user MUST read the specification. (This use of MUST indicates a keyword usage and requires an action).

2. Description and TODO

This document is to show the changes necessary to create the 1.2 version of the TCG specification. Some of the sections are brand new text; some are rewritten sections of the 1.1 version. Upon approval of the 1.2 changes, there will be a merging of the 1.1 and 1.2 versions to create a single 1.2 document.

3. Admin Startup and State

Start of informative comment:

This section is the commands that start a TPM.

End of informative comment.

3.1 TPM_Init

Start of informative comment:

TPM_Init is a physical method of initializing a TPM. There is no TPM_Init ordinal as this is a platform message sent on the platform internals to the TPM. On a PC this command arrives at the TPM via the LPC bus and informs the TPM that the platform is performing a boot process.

TPM_Init puts the TPM into a state where it waits for the command TPM_Startup (which specifies the type of initialization that is required.

End of informative comment.

Definition

```
TPM_Init();
```

Operation of the TPM. This is not a command that any software can execute. It is inherent in the design of the TPM and the platform that the TPM resides on.

Parameters

None

Description

1. The TPM_Init signal indicates to the TPM that platform initialization is taking place. The TPM SHALL set the TPM into a state such that the only legal command to receive after the TPM_Init is the TPM_Startup command. The TPM_Startup will further indicate to the TPM how to handle and initialize the TPM resources.
2. The platform design MUST be that the TPM is not the only component undergoing initialization. If the TPM_Init signal forces the TPM to perform initialization then the platform MUST ensure that ALL components of the platform receive an initialization signal. This is to prevent an attacker from causing the TPM to initialize to a state where various masquerades are allowable. For instance, on a PC causing the TPM to initialize and expect measurements in PCR0 but the remainder of the platform does not initialize.
3. The design of the TPM MUST be such that the ONLY mechanism that signals TPM_Init also signals initialization to the other platform components.

Actions

1. The TPM sets TPM_STANY_FLAGS -> postInitialise to TRUE.

3.2 TPM_Startup

Start of informative comment:

TPM_Startup is always preceded by TPM_Init, which is the physical indication (a system-wide reset) that TPM initialization is necessary.

There are many events on a platform that can cause a reset and the response to these events can require different operations to occur on the TPM. The mere reset indication does not contain sufficient information to inform the TPM as to what type of reset is occurring. Additional information known by the platform initialization code needs transmitting to the TPM. The TPM_Startup command provides the mechanism to transmit the information.

The TPM can startup in three different modes:

A "clear" start where all variables go back to their default or non-volatile set state

A "save" start where the TPM recovers appropriate information and restores various values based on a prior TPM_SaveState. This recovery requires an invocation of TPM_Init to be successful.

A failing "save" start must shut down the TPM. The CRTM cannot leave the TPM in a state where an untrusted upper software layer could issue a "clear" and then extend PCR's and thus mimic the CRTM.

A "deactivated" start where the TPM turns itself off and requires another TPM_Init before the TPM will execute in a fully operational state.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Startup
4	2	2S	2	TPM_STARTUP_TYPE	startupType	Type of startup that is occurring

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Startup

Description

TPM_Startup MUST be generated by a trusted entity (the RTM or the TPM, for example).

Actions

1. If TPM_STANY_FLAGS -> postInitialise is FALSE,
    a. Then the TPM MUST return TPM_INVALID_POSTINIT, and exit this capability
2. If stType = TPM_ST_CLEAR
    a. Ensure that sessions associated with resources TPM_RT_CONTEXT, TPM_RT_AUTH and TPM_RT_TRANS are invalidated
    b. Reset TPM_STCLEAR_DATA -> PCR[] values to each correct default value
        i. pcrReset is FALSE, set to 0x00..00
        ii. pcrReset is TRUE, set to 0xFF..FF
    c. Set the following TPM_STCLEAR_FLAGS to their default state
        i. PhysicalPresence
        ii. PhysicalPresenceLock
        iii. disableForceClear
    d. The TPM MAY initialize auditDigest to NULL
        i. If not initialized to NULL the TPM SHALL ensure that auditDigest contains a valid value
        ii. If initialization fails the TPM SHALL set auditDigest to NULL and SHALL set the internal TPM state so that the TPM returns TPM_FAILEDSELFTEST to all subsequent commands.
    e. The TPM SHALL set TPM_STCLEAR_FLAGS -> deactivated to the same state as TPM_PERMANENT_FLAGS -> deactivated
    f. The TPM MUST set the TPM_STANY_DATA fields to:
        i. TPM_STANY_DATA->contextNonceSession is set to NULLS
        ii. TPM_STANY_DATA->contextCount is set to 0
        iii. TPM_STANY_DATA->contextList is set to 0
    g. The TPM MUST set TPM_STCLEAR_DATA fields to:
        i. Invalidate contextNonceKey
        ii. countID to NULL
        iii. ownerReference to TPM_KH_OWNER
    h. The TPM MUST set the following TPM_STCLEAR_FLAGS to
        i. bGlobalLock to FALSE
    i. Determine which keys should remain in the TPM
        i. For each key that has a valid preserved value in the TPM
            (1) if parentPCRStatus is TRUE then call TPM_FlushSpecific(keyHandle)
            (2) if IsVolatile is TRUE then call TPM_FlushSpecific(keyHandle)
        ii. Keys under control of the OwnerEvict flag MUST stay resident in the TPM

3. If stType = TPM_ST_STATE
    a. If the TPM has no state to restore the TPM MUST set the internal state such that it returns TPM_FAILEDSELFTEST to all subsequent commands
    b. The TPM MAY determine for each session type (authorization, transport...) to release or maintain the session information. The TPM reports how it manages sessions in the TPM_GetCapability command.
    c. The TPM SHALL take all necessary actions to ensure that all PCRs contain valid preserved values. If the TPM is unable to successfully complete these actions, it SHALL enter the TPM failure mode.
        i. For resettable PCR the TPM MUST set the value of TPM_STCLEAR_DATA -> PCR[]to the resettable PCR default value. The TPM MUST NOT restore a resettable PCR to a preserved value
    d. The TPM MAY initialize auditDigest to NULL
        i. Otherwise, the TPM SHALL take all actions necessary to ensure that auditDigest contains a valid value. If the TPM is unable to successfully complete these actions, the TPM SHALL initialize auditDigest to NULL and SHALL set the internal set such that the TPM returns TPM_FAILEDSELFTEST to all subsequent commands.
    e. The TPM MUST restore the following flags to their preserved states:
        i. All values in TPM_STCLEAR_FLAGS
        ii. All values in TPM_STCLEAR_DATA
    f. The TPM MUST restore all keys that have a valid preserved value
    g. The TPM resumes normal operation. If the TPM is unable to resume normal operation, it SHALL enter the TPM failure mode.
4. If stType = TPM_ST_DEACTIVATED
    a. Invalidate sessions
        i. Ensure that all resources associated with saved and active sessions are invalidated
    b. Set the TPM_STCLEAR_FLAGS to their default state.
    c. Set TPM_STCLEAR_FLAGS -> deactivated to TRUE
5. The TPM MUST ensure that state associated with TPM_SaveState is invalidated
6. The TPM MUST set TPM_STANY_FLAGS -> postInitialise to FALSE
    a. postInitialize is set to FALSE even if the TPM is in failure mode.

3.3 TPM_SaveState

Start of informative comment:

This warns a TPM to save some state information.

If the relevant shielded storage is non-volatile, this command need have no effect.

If the relevant shielded storage is volatile and the TPM alone is unable to detect the loss of external power in time to move data to non-volatile memory, this command should be presented before the TPM enters a low or no power state.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveState.

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveState.

Description

1. Preserved values MUST be non-volatile.
2. If data is never stored in a volatile medium, that data MAY be used as preserved data. In such cases, no explicit action may be required to preserve that data.
3. If an explicit action is required to preserve data, it MUST be possible for the TPM to determine whether preserved data is valid.
4. If a parameter mirrored by any preserved value is altered, all preserved values MUST be declared invalid.
5. The TPM MAY declare all preserved values invalid in response to any command other than TPM_Init.

Actions

1. Store TPM_STCLEAR_DATA -> PCR contents except for
    a. If the PCR attribute pcrReset is TRUE
    b. Any platform identified debug PCR

2. The auditDigest MUST be handled according to the audit requirements as reported by TPM_GetCapability

    a. If the ordinalAuditStatus is TRUE for the TPM_SaveState ordinal and the auditDigest is being stored in the saved state, the saved auditDigest MUST include the TPM_SaveState input parameters and MUST NOT include the output parameters.

3. All values in TPM_STCLEAR_DATA MUST be preserved

4. All values in TPM_STCLEAR_FLAGS MUST be preserved

5. The contents of any key that is currently loaded SHOULD be preserved if the key's parentPCRStatus indicator is FALSE and its IsVolatile indicator is FALSE.

6. The contents of any key that has TPM_KEY_CONTROL_OWNER_EVICT set MUST be preserved

7. The contents of any key that is currently loaded MAY be preserved as reported by TPM_GetCapability

8. The contents of sessions (authorization, transport etc.) MAY be preserved as reported by TPM_GetCapability

4. Admin Testing

4.1 TPM_SelfTestFull

Start of informative comment:

TPM_SelfTestFull tests all of the TPM capabilities.

Unlike TPM_ContinueSelfTest, which may optionally return immediately and then perform the tests, TPM_SelfTestFull always performs the tests and then returns success or failure.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SelfTestFull

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SelfTestFull

Actions

1. TPM_SelfTestFull SHALL cause a TPM to perform self-test of each TPM internal function.
   a. If the self-test succeeds, return TPM_SUCCESS.
   b. If the self-test fails, return TPM_FAILEDSELFTEST.
2. Failure of any test results in overall failure, and the TPM goes into failure mode.
3. If the TPM has not executed the action of TPM_ContinueSelfTest, the TPM
   a. MAY perform the full self-test.
   b. MAY return TPM_NEEDS_SELFTEST.

4.2 TPM_ContinueSelfTest

Start of informative comment:

TPM_ContinueSelfTest informs the TPM that it should complete the self-test of all TPM functions.

The TPM may return success immediately and then perform the self-test, or it may perform the self-test and then return success or failure.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ContinueSelfTest

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ContinueSelfTest

Description

1. Prior to executing the actions of TPM_ContinueSelfTest, if the TPM receives a command C1 that uses an untested TPM function, the TPM MUST take one of these actions:
   a. The TPM MAY return TPM_NEEDS_SELFTEST
      i. This indicates that the TPM has not tested the internal resources required to execute C1.
      ii. The TPM does not execute C1.
      iii. The caller MUST issue TPM_ContinueSelfTest before re-issuing the command C1.
         (1) If the TPM permits TPM_SelfTestFull prior to completing the actions of TPM_ContinueSelfTest, the caller MAY issue TPM_SelfTestFull rather than TPM_ContinueSelfTest.
   b. The TPM MAY return TPM_DOING_SELFTEST
      i. This indicates that the TPM is doing the actions of TPM_ContinueSelfTest implicitly, as if the TPM_ContinueSelfTest command had been issued.
      ii. The TPM does not execute C1.

iii. The caller MUST wait for the actions of TPM_ContinueSelfTest to complete before reissuing the command C1.

c. The TPM MAY return TPM_SUCCESS or an error code associated with C1.

i. This indicates that the TPM has completed the actions of TPM_ContinueSelfTest and has completed the command C1.

ii. The error code MAY be TPM_FAILEDSELFTEST.

Actions

1. If TPM_PERMANENT_FLAGS -> FIPS is TRUE or TPM_PERMANENT_FLAGS -> TPMpost is TRUE

a. The TPM MUST run all self-tests

2. Else

a. The TPM MUST complete all self-tests that are outstanding

i. Instead of completing all outstanding self-tests the TPM MAY run all self-tests

3. The TPM either

a. MAY immediately return TPM_SUCCESS

i. When TPM_ContinueSelfTest finishes execution, it MUST NOT respond to the caller with a return code.

b. MAY complete the self-test and then return TPM_SUCCESS or TPM_FAILEDSELFTEST.

4.3 TPM_GetTestResult

Start of informative comment:

TPM_GetTestResult provides manufacturer specific information regarding the results of the self-test. This command will work when the TPM is in self-test failure mode. The reason for allowing this command to operate in the failure mode is to allow TPM manufacturers to obtain diagnostic information.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_GetTestResult

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_GetTestResult
4	4	3S	4	UINT32	outDataSize	The size of the outData area
5	<>	4S	<>	BYTE[]	outData	The outData this is manufacturer specific

Description

This command will work when the TPM is in self test failure mode.

Actions

1. The TPM SHALL respond to this command with a manufacturer specific block of information that describes the result of the latest self-test
2. The information MUST NOT contain any data that uniquely identifies an individual TPM.

5. Admin Opt-in

5.1 TPM_SetOwnerInstall

Start of informative comment:

When enabled but without an owner this command sets the PERMANENT flag that allows or disallows the ability to insert an owner.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOwnerInstall
4	1	2S	1	BOOL	state	State to which ownership flag is to be set.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOwnerInstall

Action

1. If the TPM has a current owner, this command immediately returns with TPM_SUCCESS.
2. The TPM validates the assertion of physical access. The TPM then sets the value of TPM_PERMANENT_FLAGS -> ownership to the value in state.

5.2 TPM_OwnerSetDisable

Start of informative comment:

The TPM owner sets the PERMANENT disable flag

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerSetDisable
4	1	2S	1	BOOL	disableState	Value for disable state – enable if TRUE
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerSetDisable
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Action

1. The TPM SHALL authenticate the command as coming from the TPM Owner. If unsuccessful, the TPM SHALL return TPM_AUTHFAIL.
2. The TPM SHALL set the TPM_PERMANENT_FLAGS -> disable flag to the value in the disableState parameter.

5.3 TPM_PhysicalEnable

Start of informative comment:

Sets the PERMANENT disable flag to FALSE using physical presence as authorization.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalEnable

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalEnable

Action

1. Validate that physical presence is being asserted, if not return TPM_BAD_PRESENCE
2. The TPM SHALL set the TPM_PERMANENT_FLAGS.disable value to FALSE.

5.4 TPM_PhysicalDisable

Start of informative comment:

Sets the PERMANENT disable flag to TRUE using physical presence as authorization

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalDisable

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_OR D_PhysicalDisable

Action

1. Validate that physical presence is being asserted, if not return TPM_BAD_PRESENCE
2. The TPM SHALL set the TPM_PERMANENT_FLAGS.disable value to TRUE.

5.5 TPM_PhysicalSetDeactivated

Start of informative comment:

Enables the TPM using physical presence as authorization.

This command is not available when the TPM is disabled.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalSetDeactivated
4	1	2S	1	BOOL	state	State to which deactivated flag is to be set.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	retumCode	The retum code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalSetDeactivated

Action

1. Validate that physical presence is being asserted, if not return TPM_BAD_PRESENCE
2. The TPM SHALL set the TPM_PERMANENT_FLAGS.deactivated flag to the value in the state parameter.

5.6 TPM_SetTempDeactivated

Start of informative comment:

This command allows the operator of the platform to deactivate the TPM until the next boot of the platform.

This command requires operator authentication. The operator can provide the authentication by either the assertion of physical presence or presenting the operator AuthData value.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetTempDeactivated
4	4		4	TPM_AUTHHANDLE	authHandle	Auth handle for operation validation. Session type MUST be OIAP
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce prev iously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	operatorAuth	HMAC key: operatorAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetTempDeactivated
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: operatorAuth.

Action

1. If tag = TPM_TAG_REQ_AUTH1_COMMAND
    a. If TPM_PERMANENT_FLAGS -> operator is FALSE return TPM_NOOPERATOR
    b. Validate command and parameters using operatorAuth, on error return TPM_AUTHFAIL
2. Else

   a. If physical presence is not asserted the TPM MUST return TPM_BAD_PRESENCE

3. The TPM SHALL set the TPM_STCLEAR_FLAGS.deactivated flag to the value TRUE.

5.7 TPM_SetOperatorAuth

Start of informative comment:

This command allows the setting of the operator AuthData value.

There is no confidentiality applied to the operator authentication as the value is sent under the assumption of being local to the platform. If there is a concern regarding the path between the TPM and the keyboard then unless the keyboard is using encryption and a secure channel an attacker can read the values.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOperatorAuth
4	20	2S	20	TPM_SECRET	operatorAuth	The operator AuthData

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	retumCode	The retum code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOperatorAuth

Action

1. If physical presence is not asserted the TPM MUST return TPM_BAD_PRESENCE
2. The TPM SHALL set the TPM_PERMANENT_DATA -> operatorAuth
3. The TPM SHALL set TPM_PERMANENT_FLAGS -> operator to TRUE

6. Admin Ownership

6.1 TPM_TakeOwnership

Start of informative comment:

This command inserts the TPM Ownership value into the TPM.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	2	2S	2	TPM_PROTOCOL_ID	protocolID	The ownership protocol in use.
5	4	3S	4	UINT32	encOwnerAuthSize	The size of the encOwnerAuth field
6	<>	4S	<>	BYTE[ ]	encOwnerAuth	The owner AuthData encrypted with PUBEK
7	4	5S	4	UINT32	encSrkAuthSize	The size of the encSrkAuth field
8	<>	6S	<>	BYTE[ ]	encSrkAuth	The SRK AuthData encrypted with PUBEK
9	<>	7S	<>	TPM_KEY	srkParams	Structure containing all parameters of new SRK. pubKey.keyLength & encSize are both 0. This structure MAY be TPM_KEY12.
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for this command
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
13	20			TPM_AUTHDATA	ownerAuth	Authorization session digest for input params. HMAC key: the new ownerAuth value. See actions for validation operations

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	<>	3S	<>	TPM_KEY	srkPub	Structure containing all parameters of new SRK. srkPub.encData is set to 0. This structure MAY be TPM_KEY12.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: the new ownerAuth value

Description

The type of the output srkPub MUST be the same as the type of the input srkParams, either both TPM_KEY or both TPM_KEY12.

Actions

1. If TPM_PERMANENT_DATA -> ownerAuth is valid return TPM_OWNER_SET
2. If TPM_PERMANENT_FLAGS -> ownership is FALSE return TPM_INSTALL_DISABLED
3. If TPM_PERMANENT_DATA -> endorsementKey is invalid return TPM_NO_ENDORSEMENT
4. Verify that authHandle is of type OIAP on error return TPM_AUTHFAIL
5. Create A1 a TPM_SECRET by decrypting encOwnerAuth using PRIVEK as the key
    a. This requires that A1 was encrypted using the PUBEK
    b. Validate that A1 is a length of 20 bytes, on error return TPM_BAD_KEY_PROPERTY
6. Validate the command and parameters using A1 and ownerAuth, on error return TPM_AUTHFAIL
7. Validate srkParams
    a. If srkParams -> keyUsage is not TPM_KEY_STORAGE return TPM_INVALID_KEYUSAGE
    b. If srkParams -> migratable is TRUE return TPM_INVALID_KEYUSAGE
    c. If srkParams -> algorithmParms -> algorithmID is NOT TPM_ALG_RSA return TPM_BAD_KEY_PROPERTY
    d. If srkParams -> algorithmParms -> encScheme is NOT TPM_ES_RSAESOAEP_SHA1_MGF1 return TPM_BAD_KEY_PROPERTY
    e. If srkParams -> algorithmParms -> sigScheme is NOT TPM_SS_NONE return TPM_BAD_KEY_PROPERTY

    f. If srkParams -> algorithmParms -> parms -> keyLength MUST be greater than or equal to 2048, on error return TPM_BAD_KEY_PROPERTY

    g. If TPM_PERMANENT_FLAGS -> FIPS is TRUE

        i. If srkParams -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

8. Generate K1 according to the srkParams on error return TPM_BAD_KEY_PROPERTY
9. Create A2 a TPM_SECRET by decrypting encSrkAuth using the PRIVEK
    a. This requires A2 to be encrypted using the PUBEK
    b. Validate that A2 is a length of 20 bytes, on error return TPM_BAD_KEY_PROPERTY
    c. Store A2 in K1 -> usageAuth
10. Store K1 in TPM_PERMANENT_DATA -> srk
11. Store A1 in TPM_PERMANENT_DATA -> ownerAuth
12. Create TPM_PERMANENT_DATA -> contextKey according to the rules for the algorithm in use by the TPM to save context blobs
13. Create TPM_PERMANENT_DATA -> delegateKey according to the rules for the algorithm in use by the TPM to save delegate blobs
14. Create TPM_PERMANENT_DATA -> tpmProof by using the TPM RNG
15. Export TPM_PERMANENT_DATA -> srk as srkPub
16. Set TPM_PERMANENT_FLAGS -> readPubek to FALSE
17. Calculate resAuth using the newly established TPM_PERMANENT_DATA -> ownerAuth

6.2 TPM_OwnerClear

Start of informative comment:

The TPM_OwnerClear command performs the clear operation under Owner authentication. This command is available until the Owner executes the TPM_DisableOwnerClear, at which time any further invocation of this command returns TPM_CLEAR_DISABLED.

All state in the TPM should be cleared when the command TPM_OwnerClear is invoked.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerClear
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Ignored
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerClear
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Fixed value FALSE
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: old ownerAuth.

Actions

1. Verify that the TPM Owner authorizes the command and all of the input, on error return TPM_AUTHFAIL.
2. If TPM_PERMANENT_FLAGS -> disableOwnerClear is TRUE then return TPM_CLEAR_DISABLED.
3. Unload all loaded keys.

4. The TPM MUST NOT modify the following TPM_PERMANENT_DATA items
    a. endorsementKey
    b. revMajor
    c. revMinor
    d. manuMaintPub
    e. auditMonotonicCounter
    f. monotonicCounter
    g. pcrAttrib
    h. rngState
    i. EKReset
    j. maxNVBufSize
    k. lastFamilyID
    l. tpmDAASeed
    m. authDIR[0]
5. The TPM MUST invalidate the following TPM_PERMANENT_DATA items and any internal resources associated with these items
    a. ownerAuth
    b. srk
    c. delegateKey
    d. delegateTable
    e. contextKey
    f. tpmProof
    g. operatorAuth
6. The TPM MUST reset to manufacturing defaults the following TPM_PERMANENT_DATA items
    a. noOwnerNVWrite MUST be set to 0
    b. ordinalAuditStatus
    c. restrictDelegate
7. The TPM MUST invalidate or reset all fields of TPM_STANY_DATA
    a. Nonces SHALL be reset
    b. Lists (e.g. contextList) SHALL be invalidated
8. The TPM MUST invalidate or reset all fields of TPM_STCLEAR_DATA
    a. Nonces SHALL be reset
    b. Lists (e.g. contextList) SHALL be invalidated
9. The TPM MUST set the following TPM_PERMANENT_FLAGS to their default values

a. disable

b. deactivated

c. readPubek

d. disableOwnerClear

10. The TPM MUST set the following TPM_PERMANENT_FLAGS

a. ownership to TRUE

b. operator to FALSE

c. maintenanceDone to FALSE

d. allowMaintenance to TRUE

11. The TPM releases all TPM_PERMANENT_DATA -> monotonicCounter settings

a. This includes invalidating all currently allocated counters. The result will be no currently allocated counters and the new owner will need to allocate counters. The actual count value will continue to increase.

12. The TPM MUST deallocate all defined NV storage areas where TPM_NV_PER_OWNERWRITE is TRUE and nvIndex does not have the "D" bit set and MUST NOT deallocate any other currently defined NV storage areas.

13. The TPM MUST invalidate all familyTable entries

14. The TPM MUST terminate all OSAP, DSAP, and transport sessions.

15. The TPM MUST terminate all sessions, active or saved

6.3 TPM_ForceClear

Start of informative comment:

The TPM_ForceClear command performs the Clear operation under physical access. This command is available until the execution of the TPM_DisableForceClear, at which time any further invocation of this command returns TPM_CLEAR_DISABLED.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ForceClear

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ForceClear

Actions

1. The TPM SHALL check for the assertion of physical presence, if not present return TPM_BAD_PRESENCE
2. If TPM_STCLEAR_FLAGS -> disableForceClear is TRUE return TPM_CLEAR_DISABLED
3. The TPM SHALL execute the actions of TPM_OwnerClear (except for the TPM Owner authentication check)

6.4 TPM_DisableOwnerClear

Start of informative comment:

The TPM_DisableOwnerClear command disables the ability to execute the TPM_OwnerClear command permanently. Once invoked the only method of clearing the TPM will require physical access to the TPM.

After the execution of TPM_ForceClear, ownerClear is re-enabled and must be explicitly disabled again by the new TPM Owner.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableOwnerClear
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableOwnerClear
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Actions

1. The TPM verifies that the authHandle properly authorizes the owner.
2. The TPM sets the TPM_PERMANENT_FLAGS -> disableOwnerClear flag to TRUE.

3. When this flag is TRUE the only mechanism that can clear the TPM is the TPM_ForceClear command. The TPM_ForceClear command requires physical access to the TPM to execute.

6.5 TPM_DisableForceClear

Start of informative comment:

The TPM_DisableForceClear command disables the execution of the TPM_ForceClear command until the next startup cycle. Once this command is executed, the TPM_ForceClear is disabled until another startup cycle is run.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableForceClear

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableForceClear

Actions

1. The TPM sets the TPM_STCLEAR_FLAGS.disableForceClear flag in the TPM that disables the execution of the TPM_ForceClear command.

6.6 TSC_PhysicalPresence

Start of informative comment:

Some TPM operations require the indication of a human's physical presence at the platform. The presence of the human either provides another indication of platform ownership or a mechanism to ensure that the execution of the command is not the result of a remote software process.

This command allows a process on the platform to indicate the assertion of physical presence. As this command is executable by software there must be protections against the improper invocation of this command.

The physicalPresenceHWEnable and physicalPresenceCMDEnable indicate the ability for either SW or HW to indicate physical presence. These flags can be reset until the physicalPresenceLifetimeLock is set. The platform manufacturer should set these flags to indicate the capabilities of the platform the TPM is bound to.

The command provides two sets of functionality. The first is to enable, permanently, either the HW or the SW ability to assert physical presence. The second is to allow SW, if enabled, to assert physical presence.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_PhysicalPresence.
4	2	2S	2	TPM_PHYSICAL_PRESENCE	physicalPresence	The state to set the TPM's Physical Presence flags.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_PhysicalPresence.

Actions

1. For documentation ease, the bits break into two categories. The first is the lifetime settings and the second is the assertion settings.
   a. Define A1 to be the lifetime settings: TPM_PHYSICAL_PRESENCE_LIFETIME_LOCK, TPM_PHYSICAL_PRESENCE_HW_ENABLE, TPM_PHYSICAL_PRESENCE_CMD_ENABLE, TPM_PHYSICAL_PRESENCE_HW_DISABLE, and TPM_PHYSICAL_PRESENCE_CMD_DISABLE

b. Define A2 to be the assertion settings: TPM_PHYSICAL_PRESENCE_LOCK, TPM_PHYSICAL_PRESENCE_PRESENT, and TPM_PHYSICAL_PRESENCE_NOTPRESENT

Lifetime lock settings

2. If any A1 setting is present

a. If TPM_PERMANENT_FLAGS -> physicalPresenceLifetimeLock is TRUE, return TPM_BAD_PARAMETER

b. If any A2 setting is present return TPM_BAD_PARAMETER

c. If both physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_ENABLE and physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_DISABLE are TRUE, return TPM_BAD_PARAMETER.

d. If both physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_ENABLE and physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_DISABLE are TRUE, return TPM_BAD_PARAMETER.

e. If physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_ENABLE is TRUE Set TPM_PERMANENT_FLAGS -> physicalPresenceHWEnable to TRUE

f. If physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_DISABLE is TRUE Set TPM_PERMANENT_FLAGS -> physicalPresenceHWEnable to FALSE

g. If physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_ENABLE is TRUE, Set TPM_PERMANENT_FLAGS -> physicalPresenceCMDEnable to TRUE.

h. If physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_DISABLE is TRUE, Set TPM_PERMANENT_FLAGS -> physicalPresenceCMDEnable to FALSE.

i. If physicalPresence -> TPM_PHYSICAL_PRESENCE_LIFETIME_LOCK is TRUE

i. Set TPM_PERMANENT_FLAGS -> physicalPresenceLifetimeLock to TRUE

j. Return TPM_SUCCESS

SW physical presence assertion

3. If any A2 setting is present

a. If any A1 setting is present return TPM_BAD_PARAMETER

i. This check here just for consistency, the prior checks would have already ensured that this was ok

b. If TPM_PERMANENT_FLAGS -> physicalPresenceCMDEnable is FALSE, return TPM_BAD_PARAMETER

c. If both physicalPresence -> TPM_PHYSICAL_PRESENCE_LOCK and physicalPresence -> TPM_PHYSICAL_PRESENCE_PRESENT are TRUE, return TPM_BAD_PARAMETER

d. If both physicalPresence -> TPM_PHYSICAL_PRESENCE_PRESENT and physicalPresence -> TPM_PHYSICAL_PRESENCE_NOTPRESENT are TRUE, return TPM_BAD_PARAMETER

e. If TPM_STCLEAR_FLAGS -> physicalPresenceLock is TRUE, return TPM_BAD_PARAMETER

```
    f. If physicalPresence -> TPM_PHYSICAL_PRESENCE_LOCK is TRUE
        i. Set TPM_STCLEAR_FLAGS -> physicalPresence to FALSE
        ii. Set TPM_STCLEAR_FLAGS -> physicalPresenceLock to TRUE
        iii. Return TPM_SUCCESS
    g. If physicalPresence -> TPM_PHYSICAL_PRESENCE_PRESENT is TRUE
        i. Set TPM_STCLEAR_FLAGS -> physicalPresence to TRUE
    h. If physicalPresence -> TPM_PHYSICAL_PRESENCE_NOTPRESENT is TRUE
        i. Set TPM_STCLEAR_FLAGS -> physicalPresence to FALSE
    i. Return TPM_SUCCESS
4. Else // There were no A1 or A2 parameters set
    a. Return TPM_BAD_PARAMETER
```

6.7 TSC_ResetEstablishmentBit

Start of informative comment:

The PC TPM Interface Specification (TIS) specifies setting tpmEstablished to TRUE upon execution of the HASH_START sequence. The setting implies the creation of a Trusted Operating System on the platform. Platforms will use the value of tpmEstablished to determine if operations necessary to maintain the security perimeter are necessary.

The tpmEstablished bit provides a non-volatile, secure reporting that a HASH_START was previously run on the platform. When a platform makes use of the tpmEstablished bit, the platform can reset tpmEstablished as the operation is no longer necessary.

For example, a platform could use tpmEstablished to ensure that, if HASH_START had ever been, executed the platform could use the value to invoke special processing. Once the processing is complete the platform will wish to reset tpmEstablished to avoid invoking the special process again.

The TPM_PERMANENT_FLAGS -> tpmEstablished bit described in the TPM specifications uses positive logic. The TPM_ACCESS register uses negative logic, so that TRUE is reflected as a 0.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_ResetEstablishmentBit

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_ResetEstablishmentBit

Actions

1. Validate the assertion of locality 3 or locality 4
2. Set TPM_PERMANENT_FLAGS -> tpmEstablished to FALSE
3. Return TPM_SUCCESS

7. The Capability Commands

Start of informative comment:

The TPM has numerous capabilities that a remote entity may wish to know about. These items include support of algorithms, key sizes, protocols and vendor-specific additions. The TPM_GetCapability command allows the TPM to report back to the requestor what type of TPM it is dealing with.

The request for information requires the requestor to specify which piece of information that is required. The request does not allow the "merging" of multiple requests and returns only a single piece of information.

In failure mode, the TPM returns a limited set of information that includes the TPM manufacturer and version.

In version 1.2 with the deletion of TPM_GetCapabilitySigned the way to obtain a signed listing of the capabilities is to create a transport session, perform TPM_GetCapability commands to list the information and then close the transport session using TPM_ReleaseTransportSigned.

End of informative comment.

1. The standard information provided in TPM_GetCapability MUST NOT provide unique information

   a. The TPM has no control of information placed into areas on the TPM like the NV store that is reported by the TPM. Configuration information for these areas could conceivably be unique

7.1 TPM_GetCapability

Start of informative comment:

This command returns current information regarding the TPM.

The limitation on what can be returned in failure mode restricts the information a manufacturer may return when capArea indicates TPM_CAP_MFR.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetCapability
4	4	2S	4	TPM_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
5	4	3S	4	UINT32	subCapSize	Size of subCap parameter
6	<>	4S	<>	BYTE[]	subCap	Further definition of information

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetCapability
4	4	3S	4	UINT32	respSize	The length of the returned capability response
5	<>	4S	<>	BYTE[ ]	resp	The capability response

Actions

1. The TPM validates the capArea and subCap indicators. If the information is available, the TPM creates the response field and fills in the actual information.
2. The structure document contains the list of caparea and subCap values
3. If the TPM is in failure mode
   a. The TPM MUST only return TPM manufacturer and TPM version.
4. If the TPM is in limited operation mode
   a. The TPM MUST only return TPM_CAP_PROPERTY -> TPM_CAP_PROP_DURATION.

7.2 TPM_SetCapability

Start of informative comment:

This command sets values in the TPM.

A setValue that is inconsistent with the capArea and subCap is considered a bad parameter.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	ordinal:TPM_ORD_SetCapability
4	4	2S	4	TPM_CAPABILITY_AREA	capArea	Partition of capabilities to be set
5	4	3S	4	UINT32	subCapSize	Size of subCap parameter
6	<>	4S	<>	BYTE[]	subCap	Further definition of information
7	4	5S	4	UINT32	setValueSize	The size of the value to set
8	<>	6S	<>	BYTE[]	setValue	The value to set
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
12	20			TPM_AUTHDATA	ownerAuth	Authorization. HMAC key: owner.usageAuth.

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal:TPM_ORD_SetCapability
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	Authorization HMAC key:owner.usageAuth.

Actions

1. If tag = TPM_TAG_RQU_AUTH1_COMMAND, validate the command and parameters using ownerAuth, return TPM_AUTHFAIL on error
2. The TPM validates the capArea and subCap indicators, including the ability to set value based on any set restrictions
3. If the capArea and subCap indicators conform with one of the entries in the structure TPM_CAPABILITY_AREA (Values for TPM_SetCapability)
    a. The TPM sets the relevant flag/data to the value of setValue parameter.
4. Else
    a. Return the error code TPM_BAD_PARAMETER.

7.3 TPM_GetCapabilityOwner

Start of informative comment:

It can provide information to TPM_GetCapabilitySigned which may result in an invalid signature.

TPM_GetCapabilityOwner enables the TPM Owner to retrieve all the non-volatile flags and the volatile flags in a single operation. This command is deprecated, mandatory.

The flags summarize many operational aspects of the TPM. The information represented by some flags is private to the TPM Owner. So, for simplicity, proof of ownership of the TPM must be presented to retrieve the set of flags. When necessary, the flags that are not private to the Owner can be deduced by Users via other (more specific) means.

The normal TPM authentication mechanisms are sufficient to prove the integrity of the response. No additional integrity check is required.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetCapbilityOwner
4	4			TPM_AUTHHANDLE	authHandle	The authorization handle used for Owner authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization digest for inputs and owner authorization. HMAC key: OwnerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation. See section 4.3.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetCapabilityOwner
4	4	3S	4	TPM_VERSION	version	A properly filled out version structure.
5	4	4S	4	UINT32	non_volatile_flags	The current state of the non-volatile flags.
6	4	5S	4	UINT32	volatile_flags	The current state of the volatile flags.
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active

9	20			TPM_AUTHDATA	resAuth	The authorization digest for the returned parameters. HMAC key: OwnerAuth.

Description

For 31>=N>=0

1. Bit-N of the TPM_PERMANENT_FLAGS structure is the Nth bit after the opening bracket in the definition of TPM_PERMANENT_FLAGS in the version of the specification indicated by the parameter "version". The bit immediately after the opening bracket is the 0th bit.
2. Bit-N of the TPM_STCLEAR_FLAGS structure is the Nth bit after the opening bracket in the definition of TPM_STCLEAR_FLAGS in the version of the specification indicated by the parameter "version". The bit immediately after the opening bracket is the 0th bit.
3. Bit-N of non_volatile_flags corresponds to the Nth bit in TPM_PERMANENT_FLAGS, and the lsb of non_volatile_flags corresponds to bit0 of TPM_PERMANENT_FLAGS
4. Bit-N of volatile_flags corresponds to the Nth bit in TPM_STCLEAR_FLAGS, and the lsb of volatile_flags corresponds to bit0 of TPM_STCLEAR_FLAGS

Actions

1. The TPM validates that the TPM Owner authorizes the command.
2. The TPM creates the parameter non_volatile_flags by setting each bit to the same state as the corresponding bit in TPM_PERMANENT_FLAGS. Bits in non_volatile_flags for which there is no corresponding bit in TPM_PERMANENT_FLAGS are set to zero.
3. The TPM creates the parameter volatile_flags by setting each bit to the same state as the corresponding bit in TPM_STCLEAR_FLAGS. Bits in volatile_flags for which there is no corresponding bit in TPM_STCLEAR_FLAGS are set to zero.
4. The TPM generates the parameter "version".
5. The TPM returns non_volatile_flags, volatile_flags and version to the caller.

8. Auditing

8.1 Audit Generation

Start of informative comment:

The TPM generates an audit event in response to the TPM executing a function that has the audit flag set to TRUE for that function.

The TPM maintains an extended value for all audited operations.

Input audit generation occurs before the listed actions and output audit generation occurs after the listed actions.

End of informative comment.

Description

1. The TPM extends the audit digest whenever the ordinalAuditStatus is TRUE for the ordinal about to be executed. The only exception is if the ordinal about to be executed is TPM_SetOrdinalAuditStatus. In that case, output parameter auditing is performed if the ordinalAuditStatus resulting from command execution is TRUE.

2. If the command is malformed

    a. If the ordinal is unknown, unimplemented, or cannot be determined, no auditing is performed.

    b. If the ordinal is known and audited, but the "above the line" parameters are malformed and the input parameter digest cannot be determined, use an input digest of all zeros.

        i. Use an output digest of the return code and ordinal.

    c. If the ordinal is known and audited, the "above the line" parameters are determined, but the "below the line" parameters are malformed, use an input digest of the "above the line" parameters.

        i. Use an output digest of the return code and ordinal.

    d. Malformed in this context means that, when breaking up a command into its parameters, there are too few or too many bytes in the command stream.

    e. Breaking up a command in this context means only the parsing required to extract the parameters.

        i. E.g., for parameter set comprising a UINT32 size and a BYTE[] array, the BYTE[] array should not be further parsed.

Actions

The TPM will execute the ordinal and perform auditing in the following manner

1. Map V1 to TPM_STANY_DATA
2. Map P1 to TPM_PERMANENT_DATA
3. If V1 -> auditDigest is NULL

```
    a. Increment P1 -> auditMonotonicCounter by 1
4. Create A1 a TPM_AUDIT_EVENT_IN structure
    a. Set A1 -> inputParms to the digest of the input parameters from the command
        i. Digest value according to the HMAC digest rules of the "above the line" parameters (i.e. the first HMAC digest calculation).
    b. Set A1 -> auditCount to P1 -> auditMonotonicCounter
    c. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A1)
5. Execute command
    a. Execution implies the performance of the listed actions for the ordinal.
6. Create A2 a TPM_AUDIT_EVENT_OUT structure
    a. Set A2 -> outputParms to the digest of the output parameters from the command
        i. Digest value according to the HMAC digest rules of the "above the line" parameters (i.e. the first HMAC digest calculation).
    b. Set A2 -> auditCount to P1 -> auditMonotonicCounter
    c. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A2)
```

8.2 Effect of audit failing after completion of a command

Start of informative comment:

An operation could complete and then when the TPM attempts to audit the command the audit process could have an internal error.

With one return parameter, The TPM is unable to return both the audit failure and the command success or failure results. To indicate the audit failure, the TPM will return one of two error codes: TPM_AUDITFAIL_SUCCESSFUL (if the command completed successfully) or TPM_AUDITFAIL_UNSUCCESSFUL (if the command completed unsuccessfully).

This new functionality changes the 1.1 TPM functionality when this condition occurs.

End of informative comment.

1. When after completion of an operation, and in performing the audit process, the TPM has an internal failure (unable to write, SHA-1 failure etc.) the TPM MUST set the internal TPM state such that the TPM returns the TPM_FAILEDSELFTEST error on subsequent attempts to execute a command
2. The return code for the command uses the following rules
    a. Command result success, Audit success -> return TPM_SUCCESS
    b. Command result failure, Audit success -> return command result failure
    c. Command result success, Audit failure -> return TPM_AUDITFAIL_SUCCESSFUL
    d. Command result failure, Audit failure -> return TPM_AUDITFAIL_UNSUCCESSFUL
3. If the TPM is permanently nonrecoverable after an audit failure, then the TPM MUST always return TPM_FAILEDSELFTEST for every command other than TPM_GetTestResult. This state must persist regardless of power cycling, the execution of TPM_Init or any other actions.

8.3 TPM_GetAuditDigest

Start of informative comment:

This returns the current audit digest. The external audit log has the responsibility to track the parameters that constitute the audit digest.

This value may be unique to an individual TPM. The value however will be changing at a rate set by the TPM Owner. Those attempting to use this value may find it changing without their knowledge. This value represents a very poor source of tracking uniqueness.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4			TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetAuditDigest
4	4			UINT32	startOrdinal	The starting ordinal for the list of audited ordinals

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4			TPM_RESULT	returnCode	The return code of the operation.
5	10			TPM_COUNTER_VALUE	counterValue	The current value of the audit monotonic counter
4	20			TPM_DIGEST	auditDigest	Log of all audited events
5	1			BOOL	more	TRUE if the output does not contain a full list of audited ordinals
5	4			UINT32	ordSize	Size of the ordinal list in bytes
6	<>			UINT32[]	ordList	List of ordinals that are audited.

Description

1. This command is never audited.

Actions

1. The TPM sets auditDigest to TPM_STANY_DATA -> auditDigest
2. The TPM sets counterValue to TPM_PERMANENT_DATA -> auditMonotonicCounter
3. The TPM creates an ordered list of audited ordinals. The list starts at startOrdinal listing each ordinal that is audited.
    a. If startOrdinal is 0 then the first ordinal that could be audited would be TPM_OIAP (ordinal 0x0000000A)

   b. The next ordinal would be TPM_OSAP (ordinal 0x0000000B)

4. If the ordered list does not fit in the output buffer the TPM sets more to TRUE
5. Return TPM_STANY_DATA -> auditDigest as auditDigest

8.4 TPM_GetAuditDigestSigned

Start of informative comment:

The signing of the audit log returns the entire digest value and the list of currently audited commands.

The inclusion of the list of audited commands as an atomic operation is to tie the current digest value with the list of commands that are being audited.

Note to future architects

When auditing functionality is active in a TPM, it may seem logical to remove this ordinal from the active set of ordinals as the signing functionality of this command could be handled in a signed transport session. While true, this command has a secondary affect also, resetting the audit log digest. As the reset requires TPM Owner authentication, there must be some way in this command to reflect the TPM Owner wishes. By requiring that a TPM Identity key be the only key that can sign and reset, the TPM Owner's authentication is implicit in the execution of the command (TPM Identity Keys are created and controlled by the TPM Owner only). Hence, while one might want to remove an ordinal this is not one that can be removed if auditing is functional.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetAuditDigestSigned
4	4			TPM_KEY_HANDLE	keyHandle	The handle of a loaded key that can perform digital signatures.
5	1	2S	1	BOOL	closeAudit	Indication if audit session should be closed
6	20	3S	20	TPM_NONCE	antiReplay	A nonce to prevent replay attacks
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for key authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	keyAuth	Authorization. HMAC key: key.usageAuth.

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetAuditDigestSigned
4	10	3S	10	TPM_COUNTER_VALUE	counterValue	The value of the audit monotonic counter
5	20	4S	20	TPM_DIGEST	auditDigest	Log of all audited events
6	20	5S	20	TPM_DIGEST	ordinalDigest	Digest of all audited ordinals
7	4	6S	4	UINT32	sigSize	The size of the sig parameter
8	<>	7S	<>	BYTE[]	sig	The signature of the area
9	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
11	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: key.usageAuth.

Actions

1. Validate the AuthData and parameters using keyAuth, return TPM_AUTHFAIL on error
2. The TPM validates that the key pointed to by keyHandle has a signature scheme of TPM_SS_RSASSAPKCS1V15_SHA1, return TPM_INVALID_KEYUSAGE on error
3. Create D1 a TPM_SIGN_INFO structure and set the structure defaults
    a. Set D1 -> fixed to "ADIG"
    b. Set D1 -> replay to antiReplay
    c. Create D3 a list of all audited ordinals as defined in the TPM_GetAuditDigest UINT32[] ordList outgoing parameter
    d. Create D4 the SHA-1 of D3
    e. Set auditDigest to TPM_STANY_DATA -> auditDigest
    f. Set counterValue to TPM_PERMANENT_DATA -> auditMonotonicCounter
    g. Create D2 the concatenation of auditDigest || counterValue || D4
    h. Set D1 -> data to D2
    i. Create a digital signature of the SHA-1 of D1 by using the signature scheme for keyHandle
    j. Set ordinalDigest to D4
4. If closeAudit == TRUE
    a. If keyHandle->keyUsage is TPM_KEY_IDENTITY
        i. TPM_STANY_DATA -> auditDigest MUST be set to NULLS.

b. Else

  i. Return TPM_INVALID_KEYUSAGE

8.5 TPM_SetOrdinalAuditStatus

Start of informative comment:

Set the audit flag for a given ordinal. Requires the authentication of the TPM Owner.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOrdinalAuditStatus
4	4	2S	4	TPM_COMMAND_CODE	ordinalToAudit	The ordinal whose audit flag is to be set
5	1	3S	1	BOOL	auditState	Value for audit flag
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes inc luding paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOrdinalAuditStatus
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Actions

1. Validate the AuthData to execute the command and the parameters
2. Validate that the ordinal points to a valid TPM ordinal, return TPM_BADINDEX on error
    a. Valid TPM ordinal means an ordinal that the TPM implementation supports
3. Set the non-volatile flag associated with ordinalToAudit to the value in auditState

9. Administrative Functions - Management

9.1 TPM_FieldUpgrade

Start of informative comment:

The TPM needs a mechanism to allow for updating the protected capabilities once a TPM is in the field. Given the varied nature of TPM implementations there will be numerous methods of performing an upgrade of the protected capabilities. This command, when implemented, provides a manufacturer specific method of performing the upgrade.

The manufacturer can determine, within the listed requirements, how to implement this command. The command may be more than one command and actually a series of commands.

The IDL definition is to create an ordinal for the command, however the remaining parameters are manufacturer specific.

The policy to determine when it is necessary to perform the actions of TPM_RevokeTrust is outside the TPM spec and determined by other TCG workgroups.

End of informative comment.

IDL Definition

```
TPM_RESULT TPM_FieldUpgrade(
   [in, out] TPM_AUTH* ownerAuth,
   …);
```

Type

This is an optional command and a TPM is not required to implement this command in any form.

Parameters

Type	Name	Description
TPM_AUTH	ownerAuth	Authentication from TPM owner to execute command
...		Remaining parameters are manufacturer specific

Descriptions

The upgrade mechanisms in the TPM MUST not require the TPM to hold a global secret. The definition of global secret is a secret value shared by more than one TPM.

The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of field upgrade. The TPM MUST NOT use the endorsement key for identification or encryption in the upgrade process. The upgrade process MAY use a TPM Identity to deliver upgrade information to specific TPM's.

The upgrade process can only change protected capabilities.

The upgrade process can only access data in shielded locations where this data is necessary to validate the TPM Owner, validate the TPME and manipulate the blob

The TPM MUST be conformant to the TPM specification, protection profiles and security targets after the upgrade. The upgrade MAY NOT decrease the security values from the original security target.

The security target used to evaluate this TPM MUST include this command in the TOE.

When a field upgrade occurs, it is always sufficient to put the TPM into the same state as a successfully executed TPM_RevokeTrust.

Actions

The TPM SHALL perform the following when executing the command:

1. Validate the TPM Owners AuthData to execute the command
2. Validate that the upgrade information was sent by the TPME. The validation mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
3. Validate that the upgrade target is the appropriate TPM model and version.
4. Process the upgrade information and update the protected capabilities
5. Set the TPM_PERMANENT_DATA.revMajor and TPM_PERMANENT_DATA.revMinor to the values indicated in the upgrade. The selection of the value is a manufacturer option. The values MUST be monotonically increasing. Installing an upgrade with a major and minor revision that is less than currently installed in the TPM is a valid operation.
6. Set the TPM_STCLEAR_FLAGS.deactivated to TRUE

9.2 TPM_SetRedirection

Informative comment

The redirection command attaches a key to a redirection receiver.

When making the connection to a GPIO channel the authorization restrictions are set at connection time and not for each invocation that uses the channel.

End of informative comments

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SetRedirection
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can implement redirection.
5	4	2S	4	TPM_REDIR_COMMAND	redirCmd	The command to execute
6	4	3S	4	UINT32	inputDataSize	The size of the input data
7	<>	4S	<>	BYTE	inputData	Manufacturer parameter
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	ownerAuth	HMAC key ownerAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetRedirection
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

Action

1. If tag == TPM_TAG_REQ_AUTH1_COMMAND

a. Validate the command and parameters using TPM Owner authentication, on error return TPM_AUTHFAIL

2. if redirCmd == TPM_REDIR_GPIO

   a. Validate that keyHandle points to a loaded key, return TPM_INVALID_KEYHANDLE on error

   b. Validate the key attributes specify redirection, return TPM_BAD_TYPE on error

   c. Validate that inputDataSize is 4, return TPM_BAD_PARAM_SIZE on error

   d. Validate that inputData points to a valid GPIO channel, return TPM_BAD_PARAMETER on error

   e. Map C1 to the TPM_GPIO_CONFIG_CHANNEL structure indicated by inputData

   f. If C1 -> attr specifies TPM_GPIO_ATTR_OWNER

      i. If tag != TPM_TAG_REQ_AUTH1_COMMAND return TPM_AUTHFAIL

   g. If C1 -> attr specifies TPM_GPIO_ATTR_PP

      i. If TPM_STCLEAR_FLAGS -> physicalPresence == FALSE, then return TPM_BAD_PRESENCE

   h. Return TPM_SUCCESS

3. The TPM MAY support other redirection types. These types may be specified by TCG or provided by the manufacturer.

9.3 TPM_ResetLockValue

Informative comment

Command that resets the TPM dictionary attack mitigation values

This allows the TPM owner to cancel the effect of a number of successive authorization failures. Dictionary attack mitigation is vendor specific, and the actions here are one possible implementation. The TPM may treat an authorization failure outside the mitigation time as a normal failure and not disable the command.

If this command itself has an authorization failure, it is blocked for the remainder of the lock out period. This prevents a dictionary attack on the owner authorization using this command.

It is understood that this command allows the TPM owner to perform a dictionary attack on other authorization values by alternating a trial and this command.

End of informative comments

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_ResetLockValue
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for TPM Owner authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	HMAC key TPM Owner auth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ResetLockValue
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	HMAC key: TPM Owner auth

Action

1. If TPM_STCLEAR_DATA -> disableResetLock is TRUE return TPM_AUTHFAIL
   a. The internal dictionary attack mechanism will set TPM_STCLEAR_DATA -> disableResetLock to FALSE when the timeout period expires
2. If the command and parameters validation using ownerAuth fails
   a. Set TPM_STCLEAR_DATA -> disableResetLock to TRUE
   b. Restart the TPM dictionary attack lock out period
   c. Return TPM_AUTHFAIL
3. Reset the internal TPM dictionary attack mitigation mechanism
   a. The mechanism is vendor specific and can include time outs, reboots, and other mitigation strategies

10. Storage functions

10.1 TPM_Seal

Start of informative comment:

The SEAL operation allows software to explicitly state the future "trusted" configuration that the platform must be in for the secret to be revealed. The SEAL operation also implicitly includes the relevant platform configuration (PCR-values) when the SEAL operation was performed. The SEAL operation uses the tpmProof value to BIND the blob to an individual TPM.

If the UNSEAL operation succeeds, proof of the platform configuration that was in effect when the SEAL operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALed secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALed, and the proof may be of no interest. On the other hand, if the SEALed secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALed. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALed. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that any SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, and the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the past SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

TPM_Seal requires the encryption of one parameter ("Secret"). For the sake of uniformity with other commands that require the encryption of more than one parameter, the string used for XOR encryption is generated by concatenating a nonce (created during the OSAP session) with the session shared secret and then hashing the result.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Seal.
4	4			TPM_KEY_HANDLE	keyHandle	Handle of a loaded key that can perform seal operations.
5	20	2S	20	TPM_ENCAUTH	encAuth	The encrypted AuthData for the sealed data. The encryption key is the shared secret from the OSAP protocol.
6	4	3S	4	UINT32	pcrInfoSize	The size of the pcrInfo parameter. If 0 there are no PCR registers in use
7	<>	4S	<>	TPM_PCR_INFO	pcrInfo	The PCR selection information. The caller MAY use TPM_PCR_INFO_LONG.
8	4	5S	4	UINT32	inDataSize	The size of the inData parameter
9	<>	6S	<>	BYTE[ ]	inData	The data to be sealed to the platform and any specified PCRs
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization. Must be an OSAP session for this command.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	Ignored
13	20			TPM_AUTHDATA	pubAuth	The authorization session digest for inputs and keyHandle. HMAC key: key.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The retum code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Seal.
4	<>	3S	<>	TPM_STORED_DATA	sealedData	Encrypted, integrity -protected data object that is the result of the TPM_Seal operation.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed value of FALSE
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the retumed parameters. HMAC key: key.usageAuth.

Descriptions

TPM_Seal is used to encrypt private objects that can only be decrypted using TPM_Unseal.

Actions

1. Validate the authorization to use the key pointed to by keyHandle

```
2. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER
3. If the keyUsage field of the key indicated by keyHandle does not have the value
   TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE.
4. If the keyHandle points to a migratable key then the TPM MUST return the error code
   TPM_INVALID_KEY_USAGE.
5. Determine the version of pcrInfo
   a. If pcrInfoSize is 0
      i. set V1 to 1
   b. Else
      i. Point X1 as TPM_PCR_INFO_LONG structure to pcrInfo
      ii. If X1 -> tag is TPM_TAG_PCR_INFO_LONG
          (1) Set V1 to 2
      iii. Else
          (1) Set V1 to 1
6. If V1 is 1 then
   a. Create S1 a TPM_STORED_DATA structure
7. else
   a. Create S1 a TPM_STORED_DATA12 structure
   b. Set S1 -> et to NULL
8. Set s1 -> encDataSize to 0
9. Set s1 -> encData to NULL
10.Set s1 -> sealInfoSize to pcrInfoSize
11.If pcrInfoSize is not 0 then
   a. if V1 is 1 then
      i. Validate pcrInfo as a valid TPM_PCR_INFO structure, return TPM_BADINDEX on
         error
      ii. Set s1 -> sealInfo -> pcrSelection to pcrInfo -> pcrSelection
      iii. Create h1 the composite hash of the PCR selected by pcrInfo -> pcrSelection
      iv. Set s1 -> sealInfo -> digestAtCreation to h1
      v. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease
   b. else
      i. Validate pcrInfo as a valid TPM_PCR_INFO_LONG structure, return
         TPM_BADINDEX on error
      ii. Set s1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection
      iii. Set s1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection
```

iv. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease

v. Set s1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease

vi. Create h2 the composite hash of the TPM_STCLEAR_DATA -> PCR selected by pcrInfo -> creationPCRSelection

vii. Set s1 -> sealInfo -> digestAtCreation to h2

viii. Set s1 -> sealInfo -> localityAtCreation to TPM_STANY_FLAGS -> localityModifier

12. If authHandle indicates XOR encryption for the AuthData secrets

a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)

b. Create a1 by XOR X1 and encAuth

13. Else

a. Create a1 by decrypting encAuth using the algorithm indicated in the OSAP session

b. Key is from authHandle -> sharedSecret

c. IV is SHA-1 of (authLastNonceEven || nonceOdd)

14. The TPM provides NO validation of a1. Well-known values (like NULLS) are valid and possible.

15. Create s2 a TPM_SEALED_DATA structure

a. Set s2 -> payload to TPM_PT_SEAL

b. Set s2 -> tpmProof to TPM_PERMANENT_DATA -> tpmProof

c. Create h3 the SHA-1 of s1

d. Set s2 -> storedDigest to h3

e. Set s2 -> authData to a1

f. Set s2 -> dataSize to inDataSize

g. Set s2 -> data to inData

16. Validate that the size of s2 can be encrypted by the key pointed to by keyHandle, return TPM_BAD_DATASIZE on error

17. Create s3 the encryption of s2 using the key pointed to by keyHandle

18. Set continueAuthSession to FALSE

19. Set s1 -> encDataSize to the size of s3

20. Set s1 -> encData to s3

21. Return s1 as sealedData

10.2 TPM_Unseal

Start of informative comment:

The TPM_Unseal operation will reveal TPM_Seal'ed data only if it was encrypted on this platform and the current configuration (as defined by the named PCR contents) is the one named as qualified to decrypt it. Internally, TPM_Unseal accepts a data blob generated by a TPM_Seal operation. TPM_Unseal decrypts the structure internally, checks the integrity of the resulting data, and checks that the PCR named has the value named during TPM_Seal. Additionally, the caller must supply appropriate AuthData for blob and for the key that was used to seal that data.

If the integrity, platform configuration and authorization checks succeed, the sealed data is returned to the caller; otherwise, an error is generated.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Unseal.
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can unseal the data.
5	<>	2S	<>	TPM_STORED_DATA	inData	The encrypted data generated by TPM_Seal.
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parentHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.
10	4			TPM_AUTHHANDLE	dataAuthHandle	The authorization session handle used to authorize inData.
		2H2	20	TPM_NONCE	dataLastNonceEven	Even nonce previously generated by TPM
11	20	3H2	20	TPM_NONCE	datanonceOdd	Nonce generated by system associated with entityAuthHandle
12	1	4H2	1	BOOL	continueDataSession	Continue usage flag for dataAuthHandle.
13	20			TPM_AUTHDATA	dataAuth	The authorization session digest for the encrypted entity. HMAC key: entity.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Unseal.
4	4	3S	4	UINT32	secretSize	The used size of the output area for secret
5	<>	4S	<>	BYTE[ ]	secret	Decrypted data that had been sealed
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.
9	20	2H2	20	TPM_NONCE	dataNonceEven	Even nonce newly generated by TPM.
		3H2	20	TPM_NONCE	datanonceOdd	Nonce generated by system associated with dataAuthHandle
10	1	4H2	1	BOOL	continueDataSession	Continue use flag, TRUE if handle is still active
11	20			TPM_AUTHDATA	dataAuth	The authorization session digest used for the dataAuth session. HMAC key: entity.usageAuth.

Actions

1. The TPM MUST validate that parentAuth authorizes the use of the key in parentHandle, on error return TPM_AUTHFAIL
2. If the keyUsage field of the key indicated by parentHandle does not have the value TPM_KEY_STORAGE, the TPM MUST return the error code TPM_INVALID_KEYUSAGE.
3. The TPM MUST check that the TPM_KEY_FLAGS -> Migratable flag has the value FALSE in the key indicated by parentHandle. If not, the TPM MUST return the error code TPM_INVALID_KEYUSAGE
4. Determine the version of inData .
    a. If inData -> tag = TPM_TAG_STORED_DATA12
        i. Set V1 to 2
        ii. Map S2 a TPM_STORED_DATA12 structure to inData
    b. Else If inData -> ver = 1.1
        i. Set V1 to 1
        ii. Map S2 a TPM_STORED_DATA structure to inData
    c. Else
        i. Return TPM_BAD_VERSION
5. Create d1 by decrypting S2 -> encData using the key pointed to by parentHandle

6. Validate d1

a. d1 MUST be a TPM_SEALED_DATA structure

b. d1 -> tpmProof MUST match TPM_PERMANENT_DATA -> tpmProof

c. Set S2 -> encDataSize to 0

d. Set S2 -> encData to NULL

e. Create h1 the SHA-1 of inData

f. d1 -> storedDigest MUST match h1

g. d1 -> payload MUST be TPM_PT_SEAL

h. Any failure MUST return TPM_NOTSEALED_BLOB

7. The TPM MUST validate authorization to use d1 by checking that the HMAC calculation using d1 -> authData as the shared secret matches the dataAuth. Return TPM_AUTHFAIL on mismatch.

8. If S2 -> sealInfoSize is not 0 then

a. If V1 is 1 then

i. Validate that S2 -> pcrInfo is a valid TPM_PCR_INFO structure

ii. Create h2 the composite hash of the PCR selected by S2 -> pcrInfo -> pcrSelection

b. If V1 is 2 then

i. Validate that S2 -> pcrInfo is a valid TPM_PCR_INFO_LONG structure

ii. Create h2 the composite hash of the TPM_STCLEAR_DATA -> PCR selected by S2 -> pcrInfo -> releasePCRSelection

iii. Check that S2 -> pcrInfo -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE

(1) For example if TPM_STANY_DATA -> localityModifier was 2 then S2 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE

c. Compare h2 with S2 -> pcrInfo -> digestAtRelease, on mismatch return TPM_WRONGPCRVAL

9. If V1 is 2 and inData -> et specifies encryption (i.e. is not NULL) then

a. If tag is not TPM_TAG_RQU_AUTH2_COMMAND, return TPM_AUTHFAIL

b. Verify that the authHandle session type is TPM_PID_OSAP, return TPM_BAD_MODE on error.

c. If inData -> et is TPM_ET_XOR

i. Use MGF1 to create string X1 of length sealedDataSize. The inputs to MGF1 are; authLastnonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The four concatenated values form the Z value that is the seed for MFG1.

ii. Create o1 by XOR of d1 -> data and X1

d. Else

i. Create o1 by encrypting d1 -> data using the algorithm indicated by inData -> et

```
        ii. Key is from authHandle -> sharedSecret
        iii. IV is SHA-1 of (authLastNonceEven || nonceOdd)
    e. Set continueAuthSession to FALSE
10. else
    a. Set o1 to d1 -> data
11. Set the return secret as o1
12. Return TPM_SUCCESS
```

10.3 TPM_UnBind

Start of informative comment:

TPM_UnBind takes the data blob that is the result of a Tspi_Data_Bind command and decrypts it for export to the User. The caller must authorize the use of the key that will decrypt the incoming blob.

TPM_UnBind operates on a block-by-block basis, and has no notion of any relation between one block and another.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_UnBind.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform UnBind operations.
5	4	2S	4	UINT32	inDataSize	The size of the input blob
6	<>	3S	<>	BYTE[ ]	inData	Encrypted blob to be decrypted
7	4			TPM_AUTHHANDLE	authHandle	The handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_UnBind
4	4	3S	4	UINT32	outDataSize	The length of the returned decrypted data
5	<>	4S	<>	BYTE[ ]	outData	The resulting decrypted data.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth.

Description

TPM_UnBind SHALL operate on a single block only.

Actions

The TPM SHALL perform the following:

1. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER
2. Validate the AuthData to use the key pointed to by keyHandle
3. If the keyUsage field of the key referenced by keyHandle does not have the value TPM_KEY_BIND or TPM_KEY_LEGACY, the TPM must return the error code TPM_INVALID_KEYUSAGE
4. Decrypt the inData using the key pointed to by keyHandle
5. if (keyHandle -> encScheme does not equal TPM_ES_RSAESOAEP_SHA1_MGF1) and (keyHandle -> keyUsage equals TPM_KEY_LEGACY),
    a. The payload does not have TPM specific markers to validate, so no consistency check can be performed.
    b. Set the output parameter outData to the value of the decrypted value of inData. (Padding associated with the encryption wrapping of inData SHALL NOT be returned.)
    c. Set the output parameter outDataSize to the size of outData, as deduced from the decryption process.
6. else
    a. Interpret the decrypted data under the assumption that it is a TPM_BOUND_DATA structure, and validate that the payload type is TPM_PT_BIND

b. Set the output parameter outData to the value of TPM_BOUND_DATA -> payloadData. (Other parameters of TPM_BOUND_DATA SHALL NOT be returned. Padding associated with the encryption wrapping of inData SHALL NOT be returned.)

c. Set the output parameter outDataSize to the size of outData, as deduced from the decryption process and the interpretation of TPM_BOUND_DATA.

7. Return the output parameters.

10.4 TPM_CreateWrapKey

Start of informative comment:

The TPM_CreateWrapKey command both generates and creates a secure storage bundle for asymmetric keys.

The newly created key can be locked to a specific PCR value by specifying a set of PCR registers.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateWrapKey
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can perform key wrapping.
5	20	2S	20	TPM_ENCAUTH	dataUsageAuth	Encrypted usage AuthData for the sealed data.
6	20	3S	20	TPM_ENCAUTH	dataMigrationAuth	Encrypted migration AuthData for the sealed data.
7	<>	4S	<>	TPM_KEY	keyInfo	Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MAY be TPM_KEY12
8	4			TPM_AUTHHANDLE	authHandle	parent key authorization. Must be an OSAP session.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Ignored
11	20			TPM_AUTHDATA	pubAuth	Authorization HMAC key: parentKey.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateWrapKey
4	<>	4S	<>	TPM_KEY	wrappedKey	The TPM_KEY structure which includes the public and encrypted private key. MAY be TPM_KEY12
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed at FALSE
7	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: parentKey.usageAuth.

Actions

The TPM SHALL do the following:

1. Validate the AuthData to use the key pointed to by parentHandle. Return TPM_AUTHFAIL on any error.
2. Validate the session type for parentHandle is OSAP.
3. If the TPM is not designed to create a key of the type requested in keyInfo, return the error code TPM_BAD_KEY_PROPERTY
4. Verify that parentHandle->keyUsage equals TPM_KEY_STORAGE
5. If parentHandle -> keyFlags -> migratable is TRUE and keyInfo -> keyFlags -> migratable is FALSE then return TPM_INVALID_KEYUSAGE
6. Validate key parameters
    a. keyInfo -> keyUsage MUST NOT be TPM_KEY_IDENTITY or TPM_KEY_AUTHCHANGE. If it is, return TPM_INVALID_KEYUSAGE
    b. If keyInfo -> keyFlags -> migrateAuthority is TRUE then return TPM_INVALID_KEYUSAGE
7. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then
    a. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS
    b. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS
    c. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS
8. If keyInfo -> keyUsage equals TPM_KEY_STORAGE or TPM_KEY_MIGRATE
        i. algorithmID MUST be TPM_ALG_RSA
        ii. encScheme MUST be TPM_ES_RSAESOAEP_SHA1_MGF1
        iii. sigScheme MUST be TPM_SS_NONE
        iv. key size MUST be 2048
9. Determine the version of key
    a. If keyInfo -> ver is 1.1
        i. Set V1 to 1
        ii. Map wrappedKey to a TPM_KEY structure
        iii. Validate all remaining TPM_KEY structures
    b. Else if keyInfo -> tag is TPM_TAG_KEY12
        i. Set V1 to 2
        ii. Map wrappedKey to a TPM_KEY12 structure
        iii. Validate all remaining TPM_KEY12 structures
10. If authHandle indicates XOR encryption for the AuthData secrets

a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)

b. Create X2 the SHA-1 of the concatenation of (authHandle -> sharedSecret || nonceOdd)

c. Create DU1 the XOR of dataUsageAuth and X1

d. Create DM1 the XOR of dataMigrationAuth and X2

11. Else

a. Decrypt dataUsageAuth and dataMigrationAuth using the algorithm indicated in the OSAP session

i. Create DU1 from dataUsageAuth

ii. Create DM1 from dataMigrationAuth

b. Key is from authHandle -> sharedSecret

c. IV is SHA-1 of (authLastNonceEven || nonceOdd)

12. Set continueAuthSession to FALSE

13. Generate asymmetric key according to algorithm information in keyInfo

14. Fill in the wrappedKey structure with information from the newly generated key.

a. Set wrappedKey -> encData -> usageAuth to DU1

b. If the KeyFlags -> migratable bit is set to 1, the wrappedKey -> encData -> migrationAuth SHALL contain the decrypted value from DataMigrationAuth.

c. If the KeyFlags -> migratable bit is set to 0, the wrappedKey -> encData -> migrationAuth SHALL be set to the value tpmProof

15. If keyInfo->PCRInfoSize is non-zero

a. If V1 is 1

i. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO structure using the pcrSelection to indicate the PCR's in use

b. Else

i. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO_LONG structure

c. Set digestAtCreation to the TPM_COMPOSITE_HASH indicated by creationPCRSelection

d. If V1 is 2 set localityAtCreation to TPM_STANY_DATA -> locality

16. Encrypt the private portions of the wrappedKey structure using the key in parentHandle

17. Return the newly generated key in the wrappedKey parameter

10.5 TPM_LoadKey2

Start of informative comment:

Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM. The TPM_LoadKey2 function loads the key into the TPM for further use.

The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle. The assumption is that the handle may change due to key management operations. It is the responsibility of upper level software to maintain the mapping between handle and any label used by external software.

This command has the responsibility of enforcing restrictions on the use of keys. For example, when attempting to load a STORAGE key it will be checked for the restrictions on a storage key (2048 size etc.).

The load command must maintain a record of whether any previous key in the key hierarchy was bound to a PCR using parentPCRStatus.

The flag parentPCRStatus enables the possibility of checking that a platform passed through some particular state or states before finishing in the current state. A grandparent key could be linked to state-1, a parent key could linked to state-2, and a child key could be linked to state-3, for example. The use of the child key then indicates that the platform passed through states 1 and 2 and is currently in state 3, in this example. TPM_Startup with stType == TPM_ST_CLEAR indicates that the platform has been reset, so the platform has not passed through the previous states. Hence keys with parentPCRStatus==TRUE must be unloaded if TPM_Startup is issued with stType == TPM_ST_CLEAR.

If a TPM_KEY structure has been decrypted AND the integrity test using "pubDataDigest" has passed AND the key is non-migratory, the key must have been created by the TPM. So there is every reason to believe that the key poses no security threat to the TPM. While there is no known attack from a rogue migratory key, there is a desire to verify that a loaded migratory key is a real key, arising from a general sense of unease about execution of arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt cycle, but this may be expensive. For RSA keys, it is therefore suggested that the consistency test consists of dividing the supposed RSA product by the supposed RSA prime, and checking that there is no remainder.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_LoadKey 2
4	4			TPM_KEY_HANDLE	parentHandle	TPM handle of parent key.
5	<>	2S	<>	TPM_KEY	inKey	Incoming key structure, both encrypted private and clear public portions. MAY be TPM_KEY12
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parentHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKey 2
4	4			TPM_KEY_HANDLE	inkeyHandle	Internal TPM handle where decrypted key was loaded.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

Actions

The TPM SHALL perform the following steps:

1. Validate the command and the parameters using parentAuth and parentHandle -> usageAuth
2. If parentHandle -> keyUsage is NOT TPM_KEY_STORAGE return TPM_INVALID_KEYUSAGE
3. If the TPM is not designed to operate on a key of the type specified by inKey, return the error code TPM_BAD_KEY_PROPERTY
4. The TPM MUST handle both TPM_KEY and TPM_KEY12 structures
5. Decrypt the inKey -> privkey to obtain TPM_STORE_ASYMKEY structure using the key in parentHandle

6. Validate the integrity of inKey and decrypted TPM_STORE_ASYMKEY

    a. Reproduce inKey -> TPM_STORE_ASYMKEY -> pubDataDigest using the fields of inKey, and check that the reproduced value is the same as pubDataDigest

7. Validate the consistency of the key and it's key usage.

    a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the public and private components of the asymmetric key pair. If inKey -> keyFlags -> migratable is FALSE, the TPM MAY verify consistency of the public and private components of the asymmetric key pair. The consistency of an RSA key pair MAY be verified by dividing the supposed (P*Q) product by a supposed prime and checking that there is no remainder..

    b. If inKey -> keyUsage is TPM_KEY_IDENTITY, verify that inKey->keyFlags->migratable is FALSE. If it is not, return TPM_INVALID_KEYUSAGE

    c. If inKey -> keyUsage is TPM_KEY_AUTHCHANGE, return TPM_INVALID_KEYUSAGE

    d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM_STORE_ASYMKEY -> migrationAuth equals TPM_PERMANENT_DATA -> tpmProof

    e. Validate the mix of encryption and signature schemes

    f. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then

        i. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS

        ii. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

        iii. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS

    g. If inKey -> keyUsage is TPM_KEY_STORAGE or TPM_KEY_MIGRATE

        i. algorithmID MUST be TPM_ALG_RSA

        ii. Key size MUST be 2048

        iii. sigScheme MUST be TPM_SS_NONE

    h. If inKey -> keyUsage is TPM_KEY_IDENTITY

        i. algorithmID MUST be TPM_ALG_RSA

        ii. Key size MUST be 2048

        iii. encScheme MUST be TPM_ES_NONE

    i. If the decrypted inKey -> pcrInfo is NULL,

        i. The TPM MUST set the internal indicator to indicate that the key is not using any PCR registers.

    j. Else

        i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a composite hash whenever the key will be in use

        ii. The TPM MUST handle both version 1.1 TPM_PCR_INFO and 1.2 TPM_PCR_INFO_LONG structures according to the type of TPM_KEY structure

            (1) The TPM MUST validate the TPM_PCR_INFO or TPM_PCR_INFO_LONG structures

8. Perform any processing necessary to make TPM_STORE_ASYMKEY key available for operations
9. Load key and key information into internal memory of the TPM. If insufficient memory exists return error TPM_NOSPACE.
10. Assign inKeyHandle according to internal TPM rules.
11. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.
12. If ParentHandle indicates that it is using PCR registers, then set inKeyHandle -> parentPCRStatus to TRUE.

10.6 TPM_GetPubKey

Start of informative comment:

The owner of a key may wish to obtain the public key value from a loaded key. This information may have privacy concerns so the command must have authorization from the key owner.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_GetPubKey.
4	4			TPM_KEY_HANDLE	keyHandle	TPM handle of key.
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	keyAuth	Authorization HMAC key: key.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_GetPubKey.
4	<>	3S	<>	TPM_PUBKEY	pubKey	Public portion of key in keyHandle.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key: key.usageAuth.

Actions

The TPM SHALL perform the following steps:

1. Validate the command the parameters using keyAuth, on error
    a. If keyHandle has TPM_AUTH_PRIV_USE_ONLY ignore the error
    b. Otherwise return TPM_AUTHFAIL
2. If keyHandle == TPM_KH_SRK then

   a. If TPM_PERMANENT_FLAGS -> readSRKPub is FALSE then return TPM_INVALID_KEYHANDLE

3. If keyHandle -> pcrInfoSize is not 0

   a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

      i. Create a digestAtRelease according to the specified PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch return TPM_WRONGPCRVAL

      ii. If specified validate any locality requests

4. Create a TPM_PUBKEY structure and return

10.7 TPM_Sealx

Start of informative comment:

The SEALX command works exactly like the SEAL command with the additional requirement of encryption for the inData parameter. This command also places in the sealed blob the information that the unseal also requires encryption.

SEALX requires the use of 1.2 data structures. The actions are the same as SEAL without the checks for 1.1 data structure usage.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SealX
4	4			TPM_KEY_HANDLE	keyHandle	Handle of a loaded key that can perform seal operations.
5	20	2S	20	TPM_ENCAUTH	encAuth	The encrypted AuthData for the sealed data. The encryption key is the shared secret from the OSAP protocol.
6	4	3S	4	UINT32	pcrInfoSize	The size of the pcrInfo parameter. If 0 there are no PCR registers in use
7	<>	4S	<>	TPM_PCR_INFO	pcrInfo	MUST use TPM_PCR_INFO_LONG.
8	4	5S	4	UINT32	inDataSize	The size of the inData parameter
9	<>	6S	<>	BYTE[ ]	inData	The data to be sealed to the platform and any specified PCRs
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization. Must be an OSAP session for this command.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	Ignored
13	20			TPM_AUTHDATA	pubAuth	The authorization session digest for inputs and keyHandle. HMAC key: key.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Sealx
4	<>	3S	4	TPM_STORED_DATA	sealedData	Encrypted, integrity -protected data object that is the result of the TPM_Sealx operation.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed value of FALSE
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth.

Actions

1. Validate the authorization to use the key pointed to by keyHandle
2. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER
3. If the keyUsage field of the key indicated by keyHandle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE.
4. If the keyHandle points to a migratable key then the TPM MUST return the error code TPM_INVALID_KEY_USAGE.
5. Create S1 a TPM_STORED_DATA12 structure
6. Set s1 -> encDataSize to 0
7. Set s1 -> encData to NULL
8. Set s1 -> sealInfoSize to pcrInfoSize
9. If pcrInfoSize is not 0 then
   a. Validate pcrInfo as a valid TPM_PCR_INFO_LONG structure, return TPM_BADINDEX on error
   b. Set s1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection
   c. Set s1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection
   d. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease
   e. Set s1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease
   f. Create h2 the composite hash of the TPM_STCLEAR_DATA -> PCR selected by pcrInfo -> creationPCRSelection
   g. Set s1 -> sealInfo -> digestAtCreation to h2
   h. Set s1 -> sealInfo -> localityAtCreation to TPM_STANY_DATA -> localityModifier
10. Create s2 a TPM_SEALED_DATA structure

11. If authHandle indicates XOR encryption for the AuthData secrets
    a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
    b. Create a1 by XOR X1 and encAuth
    c. Set s1 -> et to TPM_ET_XOR || TPM_ET_KEY
        i. TPM_ET_KEY is added because TPM_Unseal uses NULL as a special value indicating no encryption.
12. Else
    a. Create a1 by decrypting encAuth using the algorithm indicated in the OSAP session
    b. Key is from authHandle -> sharedSecret
    c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
    d. Set S1 -> et to algorithm indicated in the OSAP session
13. The TPM provides NO validation of a1. Well-known values (like NULLS) are valid and possible.
14. If authHandle indicates XOR encryption
    a. Use MGF1 to create string X2 of length inDataSize. The inputs to MGF1 are; authLastNonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The four concatenated values form the Z value that is the seed for MFG1.
    b. Create o1 by XOR of inData and X2
15. Else
    a. Create o1 by decrypting inData using the algorithm indicated by authHandle
    b. Key is from authHandle -> sharedSecret
    c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
16. Create s2 a TPM_SEALED_DATA structure
    a. Set s2 -> payload to TPM_PT_SEAL
    b. Set s2 -> tpmProof to TPM_PERMANENT_DATA -> tpmProof
    c. Create h3 the SHA-1 of s1
    d. Set s2 -> storedDigest to h3
    e. Set s2 -> authData to a1
    f. Set s2 -> dataSize to inDataSize
    g. Set s2 -> data to o1
17. Validate that the size of s2 can be encrypted by the key pointed to by keyHandle, return TPM_BAD_DATASIZE on error
18. Create s3 the encryption of s2 using the key pointed to by keyHandle
19. Set continueAuthSession to FALSE
20. Set s1 -> encDataSize to the size of s3

21. Set s1 -> encData to s3

22. Return s1 as sealedData

11. Migration

Start of informative comment:

The migration of a key from one TPM to another is a vital aspect to many use models of the TPM. The migration commands are the commands that allow this operation to occur.

There are two types of migratable keys, the version 1.1 migratable keys and the version 1.2 certifiable migratable keys.

End of informative comment.

11.1 TPM_CreateMigrationBlob

Start of informative comment:

The TPM_CreateMigrationBlob command implements the first step in the process of moving a migratable key to a new parent or platform. Execution of this command requires knowledge of the migrationAuth field of the key to be migrated.

Migrate mode is generally used to migrate keys from one TPM to another for backup, upgrade or to clone a key on another platform. To do this, the TPM needs to create a data blob that another TPM can deal with. This is done by loading in a backup public key that will be used by the TPM to create a new data blob for a migratable key.

The TPM Owner does the selection and authorization of migration public keys at any time prior to the execution of TPM_CreateMigrationBlob by performing the TPM_AuthorizeMigrationKey command.

IReWrap mode is used to directly move the key to a new parent (either on this platform or another). The TPM simply re-encrypts the key using a new parent, and outputs a normal encrypted element that can be subsequently used by a TPM_LoadKey command.

TPM_Create MigrationBlob implicitly cannot be used to migrate a non-migratory key. No explicit check is required. Only the TPM knows tpmProof. Therefore it is impossible for the caller to submit an AuthData value equal to tpmProof and migrate a non-migratory key.

End of informative comment.

483 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateMigrationBlob
4	4			TPM_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt encData.
5	2	2S	2	TPM_MIGRATE_SCHEME	migrationType	The migration type, either MIGRATE or REWRAP
6	<>	3S	<>	TPM_MIGRATIONKEYAUTH	migrationKeyAuth	Migration public key and its authorization session digest.
7	4	4S	4	UINT32	encDataSize	The size of the encData parameter
8	<>	5S	<>	BYTE[ ]	encData	The encrypted entity that is to be modified.
9	4			TPM_AUTHHANDLE	parentAuthHandle	The authorization session handle used for the parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
11	1	4H1	1	BOOL	continueAuthSession	Continue use flag for parent session
12	20		20	TPM_AUTHDATA	parentAuth	Authorization HMAC key: parentKey.usageAuth.
13	4			TPM_AUTHHANDLE	entityAuthHandle	The authorization session handle used for the encrypted entity.
		2H2	20	TPM_NONCE	entitylastNonceEven	Even nonce previously generated by TPM
14	20	3H2	20	TPM_NONCE	entitynonceOdd	Nonce generated by system associated with entityAuthHandle
15	1	4H2	1	BOOL	continueEntitySession	Continue use flag for entity session
16	20			TPM_AUTHDATA	entityAuth	Authorization HMAC key: entity.migrationAuth.

484

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateMigrationBlob
4	4	3S	4	UINT32	randomSize	The used size of the output area for random
5	<>	4S	<>	BYTE[ ]	random	String used for xor encryption
6	4	5S	4	UINT32	outDataSize	The used size of the output area for outData
7	<>	6S	<>	BYTE[ ]	outData	The modified, encrypted entity.
8	20	3H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		4H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
9	1	5H1	1	BOOL	continueAuthSession	Continue use flag for parent key session
10	20		20	TPM_AUTHDATA	resAuth	Authorization. HMAC key: parentKey.usageAuth.
11	20	3H2	20	TPM_NONCE	entityNonceEven	Even nonce newly generated by TPM to cover entity
		4H2	20	TPM_NONCE	entitynonceOdd	Nonce generated by system associated with entityAuthHandle
12	1	5 H2	1	BOOL	continueEntity Session	Continue use flag for entity session
13	20			TPM_AUTHDATA	entityAuth	Authorization HMAC key: entity.migrationAuth.

Description

The TPM does not check the PCR values when migrating values locked to a PCR.

The second authorization session (using entityAuth) MUST be OIAP because OSAP does not have a suitable entityType

Actions

1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
2. Create d1 a TPM_STORE_ASYMKEY structure by decrypting encData using the key pointed to by parentHandle.
   a. Verify that d1 -> payload is TPM_PT_ASYM.
3. Validate that entityAuth authorizes the migration of d1. The validation MUST use d1 -> migrationAuth as the secret.
4. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
5. If migrationType == TPM_MS_MIGRATE the TPM SHALL perform the following actions:
   a. Build two byte arrays, K1 and K2:
      i. K1 = d1.privKey[0..19] (d1.privKey.keyLength + 16 bytes of d1.privKey.key), sizeof(K1) = 20
      ii. K2 = d1.privKey[20..131] (position 16-127 of d1 . privKey.key), sizeof(K2) = 112
   b. Build M1 a TPM_MIGRATE_ASYMKEY structure

i. TPM_MIGRATE_ASYMKEY.payload = TPM_PT_MIGRATE

ii. TPM_MIGRATE_ASYMKEY.usageAuth = d1.usageAuth

iii. TPM_MIGRATE_ASYMKEY.pubDataDigest = d1. pubDataDigest

iv. TPM_MIGRATE_ASYMKEY.partPrivKeyLen = 112 – 127.

v. TPM_MIGRATE_ASYMKEY.partPrivKey = K2

c. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m using OAEP parameters of

i. m = M1 the TPM_MIGRATE_ASYMKEY structure

ii. pHash = d1->migrationAuth

iii. seed = s1 = K1

d. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1. Return r1 in the Random parameter.

e. Create x1 by XOR of o1 with r1

f. Copy r1 into the output field "random".

g. Encrypt x1 with the migration public key included in migrationKeyAuth.

6. If migrationType == TPM_MS_REWRAP the TPM SHALL perform the following actions:

a. Rewrap the key using the public key in migrationKeyAuth, keeping the existing contents of that key.

b. Set randomSize to 0 in the output parameter array

7. Else

a. Return TPM_BAD_PARAMETER

11.2 TPM_ConvertMigrationBlob

Start of informative comment:

This command takes a migration blob and creates a normal wrapped blob. The migrated blob must be loaded into the TPM using the normal TPM_LoadKey function.

Note that the command migrates private keys, only. The migration of the associated public keys is not specified by TPM because they are not security sensitive. Migration of the associated public keys may be specified in a platform specific specification. A TPM_KEY structure must be recreated before the migrated key can be used by the target TPM in a TPM_LoadKey command.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ConvertMigrationBlob.
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can decrypt keys.
5	4	2S	4	UINT32	inDataSize	Size of inData
6	<>	3S	<>	BYTE [ ]	inData	The XOR'd and encrypted key
7	4	4S	4	UINT32	randomSize	Size of random
8	<>	5S	<>	BYTE [ ]	random	Random value used to hide key data.
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
12	20			TPM_AUTHDATA	parentAuth	The authorization session digest that authorizes the inputs and the migration of the key in parentHandle. HMAC key: parentKey.usageAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_C OMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ConvertMigrationBlob
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[ ]	outData	The encrypted private key that can be loaded with TPM_LoadKey
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth

Action

The TPM SHALL perform the following:

1. Validate the AuthData to use the key in parentHandle
2. If the keyUsage field of the key referenced by parentHandle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE
3. Create d1 by decrypting the inData area using the key in parentHandle
4. Create o1 by XOR d1 and random parameter
5. Create m1 a TPM_MIGRATE_ASYMKEY structure, seed and pHash by OAEP decoding o1
6. Create k1 by combining seed and the TPM_MIGRATE_ASYMKEY -> partPrivKey field
7. Create d2 a TPM_STORE_ASYMKEY structure
    a. Verify that m1 -> payload == TPM_PT_MIGRATE
    b. Set d2 -> payload = TPM_PT_ASYM
    c. Set d2 -> usageAuth to m1 -> usageAuth
    d. Set d2 -> migrationAuth to pHash
    e. Set d2 -> pubDataDigest to m1 -> pubDataDigest
    f. Set d2 -> privKey field to k1
8. Create outData using the key in parentHandle to perform the encryption

11.3 TPM_AuthorizeMigrationKey

Start of informative comment:

This command creates an authorization blob, to allow the TPM owner to specify which migration facility they will use and allow users to migrate information without further involvement with the TPM owner.

It is the responsibility of the TPM Owner to determine whether migrationKey is appropriate for migration. The TPM checks just the cryptographic strength of migrationKey.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_AuthorizeMigrationKey
4	2	2S	2	TPM_MIGRATE_SCHEME	migrationScheme	Type of migration operation that is to be permitted for this key.
4	<>	3S	<>	TPM_PUBKEY	migrationKey	The public key to be authorized.
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authorization. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_AuthorizeMigrationKey
4	<>	3S	<>	TPM_MIGRATIONKEYAUTH	outData	Returned public key and authorization session digest.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Action

The TPM SHALL perform the following:

1. Check that the cryptographic strength of migrationKey is at least that of a 2048 bit RSA key. If migrationKey is an RSA key, this means that migrationKey MUST be 2048 bits or greater
2. Validate the AuthData to use the TPM by the TPM Owner
3. Create a f1 a TPM_MIGRATIONKEYAUTH structure
4. Verify that migrationKey-> algorithmParms -> encScheme is TPM_ES_RSAESOAEP_SHA1_MGF1, and return the error code TPM_INAPPROPRIATE_ENC if it is not
5. Set f1 -> migrationKey to the input migrationKey
6. Set f1 -> migrationScheme to the input migrationScheme
7. Create v1 by concatenating (migrationKey || migrationScheme || TPM_PERMANENT_DATA -> tpmProof)
8. Create h1 by performing a SHA1 hash of v1
9. Set f1 -> digest to h1
10. Return f1 as outData

11.4 TPM_MigrateKey

Start of informative comment:

The TPM_MigrateKey command performs the function of a migration authority.

The command is relatively simple; it just decrypts the input packet (coming from TPM_CreateMigrationBlob or TPM_CMK_CreateBlob) and then re-encrypts it with the input public key. The output of this command would then be sent to TPM_ConvertMigrationBlob or TPM_CMK_ConvertMigration on the target TPM.

TPM_MigrateKey does not make ANY assumptions about the contents of the encrypted blob. Since it does not have the XOR string, it cannot actually determine much about the key that is being migrated.

This command exists to permit the TPM to be a migration authority. If used in this way, it is expected that the physical security of the system containing the TPM and the AuthData value for the MA key would be tightly controlled.

To prevent the execution of this command using any other key as a parent key, this command works only if keyUsage for maKeyHandle is TPM_KEY_MIGRATE.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_MigrateKey
4	4			TPM_KEY_HANDLE	maKeyHandle	Handle of the key to be used to migrate the key.
5	<>	2S	<>	TPM_PUBKEY	pubKey	Public key to which the blob is to be migrated
6	4	3S	4	UINT32	inDataSize	The size of inData
7	<>	4S	<>	BYTE[ ]	inData	The input blob
8	4			TPM_AUTHHANDLE	maAuthHandle	The authorization session handle used for maKeyHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: maKeyHandle.usageAuth.

Outgoing Operands and Sizes

Param		HMAC		Type	Name	Description
#	Sz	#	Sz			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_MigrateKey
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[ ]	outData	The re-encrypted blob
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag for cert key session
8	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the target key. HMAC key: maKeyHandle.usageAuth

Actions

1. Validate that keyAuth authorizes the use of the key pointed to by maKeyHandle
2. The TPM validates that the key pointed to by maKeyHandle has a key usage value of TPM_KEY_MIGRATE, and that the allowed encryption scheme is TPM_ES_RSAESOAEP_SHA1_MGF1.
3. The TPM validates that pubKey is of a size supported by the TPM and that its size is consistent with the input blob and maKeyHandle.
4. The TPM decrypts inData and re-encrypts it using pubKey.

11.5 TPM_CMK_SetRestrictions

Start of informative comment:

This command is used by the Owner to dictate the usage of a certified-migration key with delegated authorization (authorization other than actual owner authorization).

This command is provided for privacy reasons and must not itself be delegated, because a certified-migration-key may involve a contractual relationship between the Owner and an external entity.

Since restrictions are validated at DSAP session use, there is no need to invalidate DSAP sessions when the restriction value changes.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_CMK_SetRestrictions
4	4	2S	4	TPM_CMK_DELEGATE	restriction	The bit mask of how to set the restrictions on CMK keys
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_CMK_SetRestrictions
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: ownerAuth.

Description

TPM_PERMANENT_DATA -> restrictDelegate is used as follows

1. If the session type is TPM_PID_DSAP and TPM_KEY -> keyFlags -> migrateAuthority is TRUE

   a. If

TPM_KEY_USAGE is TPM_KEY_SIGNING and restrictDelegate -> TPM_CMK_DELEGATE_SIGNING is TRUE, or

TPM_KEY_USAGE is TPM_KEY_STORAGE and restrictDelegate -> TPM_CMK_DELEGATE_STORAGE is TRUE, or

TPM_KEY_USAGE is TPM_KEY_BIND and restrictDelegate -> TPM_CMK_DELEGATE_BIND is TRUE, or

TPM_KEY_USAGE is TPM_KEY_LEGACY and restrictDelegate -> TPM_CMK_DELEGATE_LEGACY is TRUE, or

TPM_KEY_USAGE is TPM_KEY_MIGRATE and restrictDelegate -> TPM_CMK_DELEGATE_MIGRATE is TRUE

then the key can be used.

   b. Else return TPM_INVALID_KEYUSAGE.

Actions

1. Validate the ordinal and parameters using TPM Owner authentication, return TPM_AUTHFAIL on error
2. Set TPM_PERMANENT_DATA -> TPM_CMK_DELEGATE -> restrictDelegate = restriction
3. Return TPM_SUCCESS

11.6 TPM_CMK_ApproveMA

Start of informative comment:

This command creates an authorization ticket, to allow the TPM owner to specify which Migration Authorities they approve and allow users to create certified-migration-keys without further involvement with the TPM owner.

It is the responsibility of the TPM Owner to determine whether a particular Migration Authority is suitable to control migration

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_ApproveMA
4	20	2S	20	TPM_DIGEST	migrationAuthorityDigest	A digest of a TPM_MSA_COMPOSITE structure (itself one or more digests of public keys belonging to migration authorities)
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	Authorization HMAC, key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_CMK_ApproveMA
4	20	3S	20	TPM_HMAC	outData	HMAC of migrationAuthorityDigest
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	Authorization HMAC, key: ownerAuth.

Action

The TPM SHALL perform the following:

1. Validate the AuthData to use the TPM by the TPM Owner
2. Create M2 a TPM_CMK_MA_APPROVAL structure

    a. Set M2 ->migrationAuthorityDigest to migrationAuthorityDigest

3. Set outData = HMAC(M2) using tpmProof as the secret
4. Return TPM_SUCCESS

11.7 TPM_CMK_CreateKey

Start of informative comment:

The TPM_CMK_CreateKey command both generates and creates a secure storage bundle for asymmetric keys whose migration is controlled by a migration authority.

TPM_CMK_CreateKey is very similar to TPM_CreateWrapKey, but: (1) the resultant key must be a migratable key and can be migrated only by TPM_CMK_CreateBlob; (2) the command is Owner authorized via a ticket.

TPM_CMK_CreateKey creates an otherwise normal migratable key except that (1) migrationAuth is an HMAC of the migration authority and the new key's public key, signed by tpmProof (instead of being tpmProof); (2) the migrationAuthority bit is set TRUE; (3) the payload type is TPM_PT_MIGRATE_RESTRICTED.

The migration-selection/migration authority is specified by passing in a public key (actually the digests of one or more public keys, so more than one migration authority can be specified).

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateKey
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can perform key wrapping.
5	20	2S	20	TPM_ENCAUTH	dataUsageAuth	Encrypted usage AuthData for the sealed data.
6	<>	3S	<>	TPM_KEY12	keyInfo	Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MUST be TPM_KEY12
7	20	4S	20	TPM_HMAC	migrationAuthorityApproval	A ticket, created by the TPM Owner using TPM_CMK_ApproveMA, approving a TPM_MSA_COMPOSITE structure
8	20	5S	20	TPM_DIGEST	migrationAuthorityDigest	The digest of a TPM_MSA_COMPOSITE structure
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parent key authorization. Must be an OSAP session.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	Ignored
12	20			TPM_AUTHDATA	pubAuth	The authorization session digest that authorizes the use of the public key in parentHandle. HMAC key: parentKey.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateKey
4	<>	3S	<>	TPM_KEY12	wrappedKey	The TPM_KEY structure which includes the public and encrypted private key. MUST be TPM_KEY12
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed at FALSE
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

Actions

The TPM SHALL do the following:

1. Validate the AuthData to use the key pointed to by parentHandle. Return TPM_AUTHFAIL on any error
2. Validate the session type for parentHandle is OSAP
3. If the TPM is not designed to create a key of the type requested in keyInfo, return the error code TPM_BAD_KEY_PROPERTY
4. Verify that parentHandle->keyUsage equals TPM_KEY_STORAGE
5. Verify that parentHandle-> keyFlags-> migratable == FALSE and parentHandle-> encData -> migrationAuth == tpmProof
6. If keyInfo -> keyFlags -> migratable is FALSE, return TPM_INVALID_KEYUSAGE
7. If keyInfo -> keyFlags -> migrateAuthority is FALSE , return TPM_INVALID_KEYUSAGE
8. Verify that the migration authority is authorized
    a. Create M1 a TPM_CMK_MA_APPROVAL structure
        i. Set M1 ->migrationAuthorityDigest to migrationAuthorityDigest
    b. Verify that migrationAuthorityApproval == HMAC(M1) using tpmProof as the secret and return error TPM_MA_AUTHORITY on mismatch
9. Validate key parameters
    a. keyInfo -> keyUsage MUST NOT be TPM_KEY_IDENTITY or TPM_KEY_AUTHCHANGE. If it is, return TPM_INVALID_KEYUSAGE
10. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then
    a. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS
    b. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

   c. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS

11. If keyInfo -> keyUsage equals TPM_KEY_STORAGE or TPM_KEY_MIGRATE
    a. algorithmID MUST be TPM_ALG_RSA
    b. encScheme MUST be TPM_ES_RSAESOAEP_SHA1_MGF1
    c. sigScheme MUST be TPM_SS_NONE
    d. key size MUST be 2048
12. If keyinfo -> tag is NOT TPM_TAG_KEY12 return error TPM_INVALID_STRUCTURE
13. Map wrappedKey to a TPM_KEY12 structure
14. If authHandle indicates XOR encryption for the AuthData secrets
    a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
    b. Create DU1 by XOR X1 and dataUsageAuth
15. Else
    a. Create DU1 by decrypting dataUsageAuth using the algorithm indicated in the OSAP session
    b. Key is from authHandle -> sharedSecret
    c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
16. Set continueAuthSession to FALSE
17. Generate asymmetric key according to algorithm information in keyInfo
18. Fill in the wrappedKey structure with information from the newly generated key.
    a. Set wrappedKey -> encData -> usageAuth to DU1
    b. Set wrappedKey -> encData -> payload to TPM_PT_MIGRATE_RESTRICTED
    c. Create thisPubKey, a TPM_PUBKEY structure containing wrappedKey's public key and algorithm parameters
    d. Create M2 a TPM_CMK_MIGAUTH structure
        i. Set M2 -> msaDigest to migrationAuthorityDigest
        ii. Set M2 -> pubKeyDigest to SHA-1 (thisPubKey)
    e. Set wrappedKey -> encData -> migrationAuth equal to HMAC(M2), using tpmProof as the shared secret
19. If wrappedKey->PCRInfoSize is non-zero
    a. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO_LONG structure
    b. Set digestAtCreation to the TPM_COMPOSITE_HASH indicated by creationPCRSelection
    c. Set localityAtCreation to TPM_STANY_FLAGS -> localityModifier
20. Encrypt the private portions of the wrappedKey structure using the key in parentHandle
21. Return the newly generated key in the wrappedKey parameter

11.8 TPM_CMK_CreateTicket

Start of informative comment:

The TPM_CMK_CreateTicket command uses a public key to verify the signature over a digest.

TPM_CMK_CreateTicket returns a ticket that can be used to prove to the same TPM that signature verification with a particular public key was successful.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateTicket
4	<>	2S	<>	TPM_PUBKEY	verificationKey	The public key to be used to check signatureValue
5	20	3S	20	TPM_DIGEST	signedData	The data to be verified
6	4	4S	4	UINT32	signatureValueSize	The size of the signatureValue
7	<>	5S	<>	BYTE[]	signatureValue	The signatureValue to be verified
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Ignored
11	20			TPM_AUTHDATA	pubAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateTicket
4	20	3S	20	TPM_HMAC	sigTicket	Ticket that proves digest created on this TPM
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag
7	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key:. ownerAuth.

Actions

The TPM SHALL do the following:

1. Validate the TPM Owner authentication to use the command
2. Validate that the key type and algorithm are correct
    a. Validate that verificationKey -> algorithmParms -> algorithmID == TPM_ALG_RSA
    b. Validate that verificationKey -> algorithmParms ->encScheme == TPM_ES_NONE
    c. Validate that verificationKey ->algorithmParms ->sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1
3. Use verificationKey to verify that signatureValue is a valid signature on signedData, and return error TPM_BAD_SIGNATURE on mismatch
4. Create M2 a TPM_CMK_SIGTICKET
    a. Set M2 -> verKeyDigest to the SHA-1 (verificationKey)
    b. Set M2 -> signedData to signedData
5. Set sigTicket = HMAC(M2) signed by using tpmProof as the secret
6. Return TPM_SUCCESS

11.9 TPM_CMK_CreateBlob

Start of informative comment:

TPM_CMK_CreateBlob command is very similar to TPM_CreateMigrationBlob, except that it: (1) uses an extra ticket (restrictedKeyAuth) instead of a migrationAuth authorization session; (2) uses the migration options TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE_DOUBLE; (3) produces a wrapped key blob whose migrationAuth is independent of tpmProof.

If the destination (parent) public key is the MA, migration is implicitly permitted. Further checks are required if the MA is not the destination (parent) public key, and merely selects a migration destination: (1) sigTicket must prove that restrictTicket was signed by the MA; (2) restrictTicket must vouch that the target public key is approved for migration to the destination (parent) public key. (Obviously, this more complex method may also be used by an MA to approve migration to that MA.) In both cases, the MA must be one of the MAs implicitly listed in the migrationAuth of the target key-to-be-migrated.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_OR D_CMK_CreateBlob
4	4			TPM_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt encData.
5	2	2S	2	TPM_MIGRATE_SCHEME	migrationType	The migration type, either TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE_DOUBLE
6	<>	3S	<>	TPM_MIGRATIONKEYAUTH	migrationKeyAuth	Migration public key and its authorization session digest.
7	20	4S	20	TPM_DIGEST	pubSourceKeyDigest	The digest of the TPM_PUBKEY of the entity to be migrated
8	4	5S	4	UINT32	msaListSize	The size of the msaList parameter, which is a variable length TPM_MSA_COMPOSITEstructure
9	<>	6S	<>	TPM_MSA_COMPOSITE	msaList	One ormore digests of public keys belonging to migration authorities
10	4	7S	4	UINT32	restrictTicketSize	The size of the restrictTicket parameter, which is a TPM_CMK_AUTH structure if migration type is TPM_MS_RESTRICT_APPROVE_DOUBLE
11	<>	8S	<>	BYTE[]	restrictTicket	Either a NULL parameter or a TPM_CMK_AUTH structure, containing the digests of the public key s belonging to the Migration Authority, the destination parent key and the key -to-be-migrated.
12	4	9S	4	UINT32	sigTicketSize	The size of the sigTicket parameter, which is a TPM_HMAC structure if migration type is TPM_MS_RESTRICT_APPROVE_DOUBLE.
13	<>	10S	<>	BYTE[]	sigTicket	Either a NULL parameter or a TPM_HMAC structure, generated by the TPM, signaling a valid signature over restrictTicket
14	4	11S	4	UINT32	encDataSize	The size of the encData parameter
15	<>	12S	<>	BYTE[]	encData	The encrypted entity that is to be modified.
16	4			TPM_AUTHHANDLE	parentAuthHandle	The authorization session handle used for the parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
17	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
18	1	4H1	1	BOOL	continueAuthSession	Continue use flag for parent session
19	20		20	TPM_AUTHDATA	parentAuth	HMAC key: parentKey.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateBlob
4	4	3S	4	UINT32	randomSize	The used size of the output area for random
5	<>	4S	<>	BYTE[ ]	random	String used for xor encryption
6	4	5S	4	UINT32	outDataSize	The used size of the output area for outData
7	<>	6S	<>	BYTE[ ]	outData	The modified, encrypted entity.
8	20	3H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		4H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
9	1	5H1	1	BOOL	continueAuthSession	Continue use flag for parent key session
10	20		20	TPM_AUTHDATA	resAuth	HMAC key: parentKey.usageAuth.

Description

The TPM does not check the PCR values when migrating values locked to a PCR.

Actions

1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
2. Verify that parentHandle-> keyFlags-> migratable == FALSE and parentHandle-> encData -> migrationAuth == tpmProof
3. Create d1 by decrypting encData using the key pointed to by parentHandle.
4. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
5. Verify that d1 -> payload == TPM_PT_MIGRATE_RESTRICTED or TPM_PT_MIGRATE_EXTERNAL
6. Verify that the migration authorities in msaList are authorized to migrate this key

   a. Create M2 a TPM_CMK_MIGAUTH structure

      i. Set M2 -> msaDigest to SHA1[msaList]

      ii. Set M2 -> pubKeyDigest to pubSourceKeyDigest

   b. Verify that d1 -> migrationAuth == HMAC(M2) using tpmProof as the secret and return error TPM_MA_AUTHORITY on mismatch

7. If migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_MIGRATE

   a. Verify that intended migration destination is an MA:

      i. For one of n=1 to n=(msaList -> MSAlist), verify that SHA1[migrationKeyAuth -> migrationKey] == msaList -> migAuthDigest[n]

   b. Validate that the MA key is the correct type

i. Validate that migrationKeyAuth -> migrationKey -> algorithmParms -> algorithmID == TPM_ALG_RSA

ii. Validate that migrationKeyAuth -> migrationKey -> algorithmParms -> encScheme is an encryption scheme supported by the TPM

iii. Validate that migrationKeyAuth -> migrationKey ->algorithmParms -> sigScheme is TPM_SS_NONE

8. else If migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_APPROVE_DOUBLE,

a. Verify that the intended migration destination has been approved by the MSA:

i. Verify that for one of the n=1 to n=(msaList -> MSAlist) values of msaList -> migAuthDigest[n], sigTicket == HMAC (V1) using tpmProof as the secret where V1 is a TPM_CMK_SIGTICKET structure such that:

(1) V1 -> verKeyDigest = msaList -> migAuthDigest[n]

(2) V1 -> signedData = SHA1[restrictTicket]

ii. If [restrictTicket -> destinationKeyDigest] != SHA1[migrationKeyAuth -> migrationKey], return error TPM_MA_DESTINATION

iii. If [restrictTicket -> sourceKeyDigest] != pubSourceKeyDigest, return error TPM_MA_SOURCE

9. Else return with error TPM_BAD_PARAMETER.

10. Build two bytes array, K1 and K2, using d1:

a. K1 = TPM_STORE_ASYMKEY.privKey[0..19] (TPM_STORE_ASYMKEY.privKey.keyLength + 16 bytes of TPM_STORE_ASYMKEY.privKey.key), sizeof(K1) = 20

b. K2 = TPM_STORE_ASYMKEY.privKey[20..131] (position 16-127 of TPM_STORE_ASYMKEY . privKey.key), sizeof(K2) = 112

11. Build M1 a TPM_MIGRATE_ASYMKEY structure

a. TPM_MIGRATE_ASYMKEY.payload = TPM_PT_CMK_MIGRATE

b. TPM_MIGRATE_ASYMKEY.usageAuth = TPM_STORE_ASYMKEY.usageAuth

c. TPM_MIGRATE_ASYMKEY.pubDataDigest = TPM_STORE_ASYMKEY. pubDataDigest

d. TPM_MIGRATE_ASYMKEY.partPrivKeyLen = 112 – 127.

e. TPM_MIGRATE_ASYMKEY.partPrivKey = K2

12. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m using OAEP parameters m, pHash, and seed

a. m is the previously created M1

b. pHash = SHA1(SHA1[msaList] || pubSourceKeyDigest)

c. seed = s1 = the previously created K1

13. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1. Return r1 in the random parameter

14. Create x1 by XOR of o1 with r1
15. Copy r1 into the output field “random”
16. Encrypt x1 with the migrationKeyAuth-> migrationKey

11.10TPM_CMK_ConvertMigration

Start of informative comment:

TPM_CMK_ConvertMigration completes the migration of certified migration blobs.

This command takes a certified migration blob and creates a normal wrapped blob with payload type TPM_PT_MIGRATE_EXTERNAL. The migrated blob must be loaded into the TPM using the normal TPM_LoadKey function.

Note that the command migrates private keys, only. The migration of the associated public keys is not specified by TPM because they are not security sensitive. Migration of the associated public keys may be specified in a platform specific specification. A TPM_KEY structure must be recreated before the migrated key can be used by the target TPM in a TPM_LoadKey command.

TPM_CMK_ConvertMigration checks that one of the MAs implicitly listed in the migrationAuth of the target key has approved migration of the target key to the destination (parent) key, and that the settings (flags etc.) in the target key are those of a CMK.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_ConvertMigration
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can decrypt keys.
5	60	2S	60	TPM_CMK_AUTH	restrictTicket	The digests of public keys belonging to the Migration Authority, the destination parent key and the key -to-be-migrated.
6	20	3S	20	TPM_HMAC	sigTicket	A signature ticket, generated by the TPM, signaling a valid signature over restrictTicket
7	<>	4S	<>	TPM_KEY12	migratedKey	The public key of the key -to-be-migrated. The private portion MUST be TPM_MIGRATE_ASYMKEY properly XOR'd
8	4	5S	4	UINT32	msaListSize	The size of the msaList parameter, which is a variable length TPM_MSA_COMPOSITEstructure
9	<>	6S	<>	TPM_MSA_COMPOSITE	msaList	One or more digests of public keys belonging to migration authorities
10	4	7S	4	UINT32	randomSize	Size of random
11	<>	8S	<>	BYTE [ ]	random	Random value used to hide key data.
12	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
13	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
14	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
15	20			TPM_AUTHDATA	parentAuth	Authorization HMAC : parentKey.usageAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_ConvertMigration
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[ ]	outData	The encrypted private key that can be loaded with TPM_LoadKey
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	Authorization HMAC key .usageAuth

Action

1. Validate the AuthData to use the key in parentHandle
2. If the keyUsage field of the key referenced by parentHandle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE
3. Create d1 by decrypting the migratedKey -> encData area using the key in parentHandle
4. Create o1 by XOR d1 and random parameter
5. Create m1 a TPM_MIGRATE_ASYMKEY, seed and pHash by OAEP decoding o1
6. Create migratedPubKey a TPM_PUBKEY structure corresponding to migratedKey
    a. Verify that pHash == SHA1(SHA1[msaList] | | SHA1(migratedPubKey)
7. Create k1 by combining seed and the TPM_MIGRATE_ASYMKEY -> partPrivKey field
8. Create d2 a TPM_STORE_ASYMKEY structure.
    a. Set the TPM_STORE_ASYMKEY -> privKey field to k1
    b. Set d2 -> usageAuth to m1 -> usageAuth
    c. Set d2 -> pubDataDigest to m1 -> pubDataDigest
9. Verify that parentHandle-> keyFlags -> migratable == FALSE and parentHandle-> encData -> migrationAuth == tpmProof
10. Verify that m1 -> payload == TPM_PT_CMK_MIGRATE then set d2-> payload = TPM_PT_MIGRATE_EXTERNAL
11. Verify that for one of the n=1 to n=(msaList -> MSAlist) values of msaList -> migAuthDigest[n] sigTicket == HMAC (V1) using tpmProof as the secret where V1 is a TPM_CMK_SIGTICKET structure such that:
    a. V1 -> verKeyDigest = msaList -> migAuthDigest[n]
    b. V1 -> signedData = SHA1[restrictTicket]

12. Create parentPubKey, a TPM_PUBKEY structure corresponding to parentHandle
13. If [restrictTicket -> destinationKeyDigest] != SHA1(parentPubKey), return error TPM_MA_DESTINATION
14. Verify that migratedKey is cor responding to d2
15. If migratedKey -> keyFlags -> migratable is FALSE, and return error TPM_INVALID_KEYUSAGE
16. If migratedKey -> keyFlags -> migrateAuthority is FALSE, return error TPM_INVALID_KEYUSAGE
17. If [restrictTicket -> sourceKeyDigest] != SHA1(migratedPubKey), return error TPM_MA_SOURCE
18. Create M2 a TPM_CMK_MIGAUTH structure
    a. Set M2 -> msaDigest to SHA1[msaList]
    b. Set M2 -> pubKeyDigest to SHA1[migratedPubKey]
19. Set d2 -> migrationAuth = HMAC(M2) using tpmProof as the secret
20. Create outData using the key in parentHandle to perform the encryption

12. Maintenance Functions (optional)

Start of informative comment:

When a maintenance archive is created with generateRandom FALSE, the maintenance blob is XOR encrypted with the owner authorization before encryption with the maintenance public key. This prevents the manufacturer from obtaining plaintext data. The receiving TPM must have the same owner authorization as the sending TPM in order to XOR decrypt the archive.

When generateRandom is TRUE, the maintenance blob is XOR encrypted with random data, which is also returned. This permits someone trusted by the Owner to load the maintenance archive into the replacement platform in the absence of the Owner and manufacturer, without the Owner having to reveal information about his auth value. The receiving and sending TPM's may have different owner authorizations. The random data is transferred from the sending TPM owner to the receiving TPM owner out of band, so the maintenance blob remains hidden from the manufacturer.

This is a typical maintenance sequence:

1. Manufacturer:

- generates maintenance key pair
- gives public key to TPM1 owner

2. TPM1: TPM_LoadManuMaintPub

- load maintenance public key

3. TPM1: TPM_CreateMaintenanceArchive

- XOR encrypt with owner auth or random
- encrypt with maintenance public key

4. Manufacturer:

- decrypt with maintenance private key
- (still XOR encrypted with owner auth or random)
- encrypt with TPM2 SRK public key

5. TPM2: TPM_LoadMaintenanceArchive

- decrypt with SRK private key
- XOR decrypt with owner auth or random

End of informative comment.

12.1 TPM_CreateMaintenanceArchive

Start of informative comment:

This command creates the maintenance archive. It can only be executed by the owner, and may be shut off with the TPM_KillMaintenanceFeature command.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Cmd ordinal: TPM_ORD_CreateMaintenanceArchive
4	1	2S	1	BOOL	generateRandom	Use RNG or Owner auth to generate 'random'.
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Ev en nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Cmd ordinal: TPM_ORD_CreateMaintenanceArchive
4	4	3S	4	UINT32	randomSize	Size of the returned random data. Will be 0 if generateRandom is FALSE.
5	<>	4S	<>	BYTE [ ]	random	Random data to XOR with result.
6	4	5S	4	UINT32	archiveSize	Size of the encrypted archive
7	<>	6S	<>	BYTE [ ]	archive	Encrypted key archive.
8	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Actions

Upon authorization being confirmed this command does the following:

1. Validates that the TPM_PERMANENT_FLAGS -> allowMaintenance is TRUE. If it is FALSE, the TPM SHALL return TPM_DISABLED_CMD and exit this capability.
2. Validates the TPM Owner AuthData.
3. If the value of TPM_PERMANENT_DATA -> manuMaintPub is zero, the TPM MUST return the error code TPM_KEYNOTFOUND
4. Build a1 a TPM_KEY structure using the SRK. The encData field is not a normal TPM_STORE_ASYMKEY structure but rather a TPM_MIGRATE_ASYMKEY structure built using the following actions.
5. Build a TPM_STORE_PRIVKEY structure from the SRK. This privKey element should be 132 bytes long for a 2K RSA key.
6. Create k1 and k2 by splitting the privKey element created in step 4 into 2 parts. k1 is the first 20 bytes of privKey, k2 contains the remainder of privKey.
7. Build m1 by creating and filling in a TPM_MIGRATE_ASYMKEY structure
    a. m1 -> usageAuth is set to TPM_PERMANENT_DATA -> tpmProof
    b. m1 -> pubDataDigest is set to the digest value of the SRK fields from step 4
    c. m1 -> payload is set to TPM_PT_MAINT
    d. m1 -> partPrivKey is set to k2
8. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m using OAEP parameters of
    a. m = TPM_MIGRATE_ASYMKEY structure (step 7)
    b. pHash = TPM_PERMANENT_DATA -> ownerAuth
    c. seed = s1 = k1 (step 6)
9. If generateRandom = TRUE
    a. Create r1 by obtaining values from the TPM RNG. The size of r1 MUST be the same size as o1. Set random parameter to r1
10. If generateRandom = FALSE
    a. Create r1 by applying MGF1 to the TPM Owner AuthData. The size of r1 MUST be the same size as o1. Set random parameter to null.
11. Create x1 by XOR of o1 with r1
12. Encrypt x1 with the manuMaintPub key using the TPM_ES_RSAESOAEP_SHA1_MGF1 encryption scheme.
13. Set a1 -> encData to the encryption of x1
14. Set TPM_PERMANENT_FLAGS -> maintenanceDone to TRUE
15. Return a1 in the archive parameter

12.2 TPM_LoadMaintenanceArchive

Start of informative comment:

This command loads in a Maintenance archive that has been massaged by the manufacturer to load into another TPM.

If the maintenance archive was created using the owner authorization for XOR encryption, the current owner authorization must be used for decryption. The owner authorization does not change.

If the maintenance archive was created using random data for the XOR encryption, the vendor specific arguments must include the random data. The owner authorization may change.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadMaintenanceArchive
4	4	2S	4	UINT32	archiveSize	Sice of the encrypted archive
5	<>	3S	<>	BYTE[]	archive	Encrypted key archive
				...	...	Vendor specific arguments
-	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
			20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
-	20		20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
-	1		1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
--	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4		4	TPM_RESULT	returnCode	The return code of the operation.
			4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadMaintenanceArchive
				..	..	Vendor specific arguments
-	20		20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
			20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
-	1		1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
-	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth, the original value and not the new auth value

Descriptions

The maintenance mechanisms in the TPM MUST not require the TPM to hold a global secret. The definition of global secret is a secret value shared by more than one TPM.

The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of maintenance. The TPM MUST NOT use the endorsement key for identification or encryption in the maintenance process. The maintenance process MAY use a TPM Identity to deliver maintenance information to specific TPM's.

The maintenance process can only change the SRK, tpmProof and TPM Owner AuthData fields.

The maintenance process can only access data in shielded locations where this data is necessary to validate the TPM Owner, validate the TPME and manipulate the blob

The TPM MUST be conformant to the TPM specification, protection profiles and security targets after maintenance. The maintenance MAY NOT decrease the security values from the original security target.

The security target used to evaluate this TPM MUST include this command in the TOE.

Actions

The TPM SHALL perform the following when executing the command

1. Validate the TPM Owner's AuthData
2. Validate that the maintenance information was sent by the TPME. The validation mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
3. The packet MUST contain m2 as defined in section 12.1.
4. Ensure that only the target TPM can interpret the maintenance packet. The protection mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
5. Process the maintenance information

  a. Update the SRK
     i. Set the SRK usageAuth to be the same as the source TPM owner's AuthData
  b. Update TPM_PERMANENT_DATA -> tpmProof
  c. Update TPM_PERMANENT_DATA -> ownerAuth

6. Set TPM_PERMANENT_FLAGS -> maintenanceDone to TRUE
7. Terminate all OSAP and DSAP sessions

12.3 TPM_KillMaintenanceFeature

Informative Comments:

The TPM_KillMaintencanceFeature is a permanent action that prevents ANYONE from creating a maintenance archive. This action, once taken, is permanent until a new TPM Owner is set.

This action is to allow those customers who do not want the maintenance feature to not allow the use of the maintenance feature.

At the discretion of the Owner, it should be possible to kill the maintenance feature in such a way that the only way to recover maintainability of the platform would be to wipe out the root keys. This feature is mandatory in any TPM that implements the maintenance feature.

End informative Comment

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total num ber of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_KillMaintenanceFeature
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_KillMaintenanceFeature
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	HMAC key: ownerAuth.

Actions

1. Validate the TPM Owner AuthData
2. Set the TPM_PERMANENT_FLAGS.allowMaintenance flag to FALSE.

12.4 TPM_LoadManuMaintPub

Informative Comments:

The TPM_LoadManuMaintPub command loads the manufacturer's public key for use in the maintenance process. The command installs manuMaintPub in PERMANENT data storage inside a TPM. Maintenance enables duplication of non-migratory data in protected storage. There is therefore a security hole if a platform is shipped before the maintenance public key has been installed in a TPM.

The command is expected to be used before installation of a TPM Owner or any key in TPM protected storage. It therefore does not use authorization.

End of Informative Comments

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadManuMaintPub
4	20	2S	20	TPM_NONCE	antiReplay	AntiReplay and validation nonce
5	<>	3S	<>	TPM_PUBKEY	pubKey	The public key of the manufacturer to be in use for maintenance

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadManuMaintPub
4	20	3S	20	TPM_DIGEST	checksum	Digest of pubKey and antiReplay

Description

The pubKey MUST specify an algorithm whose strength is not less than the RSA algorithm with 2048bit keys.

pubKey SHOULD unambiguously identify the entity that will perform the maintenance process with the TPM Owner.

TPM_PERMANENT_DATA -> manuMaintPub SHALL exist in a TPM-shielded location, only.

If an entity (Platform Entity) does not support the maintenance process but issues a platform credential for a platform containing a TPM that supports the maintenance process, the value of TPM_PERMANENT_DATA -> manuMaintPub MUST be set to zero before the platform leaves the entity's control. That is, this ordinal can only be run once, and used to either load the key or load a NULL key.

Actions

The first valid TPM_LoadManuMaintPub command received by a TPM SHALL

1. Store the parameter pubKey as TPM_PERMANENT_DATA -> manuMaintPub.
2. Set checksum to SHA-1 of (pubKey || antiReplay)
3. Export the checksum
4. Subsequent calls to TPM_LoadManuMaintPub SHALL return code TPM_DISABLED_CMD.

12.5 TPM_ReadManuMaintPub

Informative Comments:

The TPM_ReadManuMaintPub command is used to check whether the manufacturer's public maintenance key in a TPM has the expected value. This may be useful during the manufacture process. The command returns a digest of the installed key, rather than the key itself. This hinders discovery of the maintenance key, which may (or may not) be useful for manufacturer privacy.

The command is expected to be used before installation of a TPM Owner or any key in TPM protected storage. It therefore does not use authorization.

End of Informative Comments

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadManuMaintPub
4	20	2S	20	TPM_NONCE	antiReplay	AntiReplay and validation nonce

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadManuMaintPub
4	20	3S	20	TPM_DIGEST	checksum	Digest of pubKey and antiReplay

Description

This command returns the hash of the antiReplay nonce and the previously loaded manufacturer's maintenance public key.

Actions

The TPM_ReadManuMaintPub command SHALL

1. Create "checksum" by concatenating data to form (TPM_PERMANENT_DATA -> manuMaintPub ||antiReplay) and passing the concatenated data through SHA1.
2. Export the checksum

13. Cryptographic Functions

13.1 TPM_SHA1Start

Start of informative comment:

This capability starts the process of calculating a SHA-1 digest.

The exposure of the SHA-1 processing is a convenience to platforms in a mode that do not have sufficient memory to perform SHA-1 themselves. As such, the use of SHA-1 is restrictive on the TPM.

The TPM may not allow any other types of processing during the execution of a SHA-1 session. There is only one SHA-1 session active on a TPM.

After the execution of TPM_SHA1Start, and prior to TPM_SHA1Complete or TPM_SHA1CompleteExtend, the receipt of any command other than TPM_SHA1Update will cause the invalidation of the SHA-1 session.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SHA1Start

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SHA1Start
4	4	3S	4	UINT32	maxNumBytes	Maximum number of bytes that can be sent to TPM_SHA1Update. Must be a multiple of 64 bytes.

Description

1. This capability prepares the TPM for a subsequent TPM_SHA1Update, TPM_SHA1Complete or TPM_SHA1CompleteExtend command. The capability SHALL open a thread that calculates a SHA-1 digest.

2. After receipt to TPM_SHA1Start, and prior to the receipt of TPM_SHA1Complete or TPM_SHA1CompleteExtend, receipt of any command other than TPM_SHA1Update invalidates the SHA-1 session.

   a. If the command received is TPM_ExecuteTransport, the SHA-1 session invalidation is based on the wrapped command, not the TPM_ExecuteTransport ordinal.

13.2 TPM_SHA1Update

Start of informative comment:

This capability inputs complete blocks of data into a pending SHA-1 digest. At the end of the process, the digest remains pending.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SHA1Update
4	4	2S	4	UINT32	numBytes	The number of bytes in hashData. Must be a multiple of 64 bytes.
5	<>	3S	<>	BYTE [ ]	hashData	Bytes to be hashed

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SHA1Update

Description

This command SHALL incorporate complete blocks of data into the digest of an existing SHA-1 thread. Only integral numbers of complete blocks (64 bytes each) can be processed.

13.3 TPM_SHA1Complete

Start of informative comment:

This capability terminates a pending SHA-1 calculation.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SHA1Complete
4	4	2S	4	UINT32	hashDataSize	Number of bytes in hashData, MUST be 64 or less
5	<>	3S	<>	BYTE [ ]	hashData	Final bytes to be hashed

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SHA1Complete
4	20	3S	20	TPM_DIGEST	hashValue	The output of the SHA-1 hash.

Description

This command SHALL incorporate a partial or complete block of data into the digest of an existing SHA-1 thread, and terminate that thread. hashDataSize MAY have values in the range of 0 through 64, inclusive.

If the SHA-1 thread has received no bytes the TPM SHALL calculate the SHA-1 of the empty buffer.

13.4 TPM_SHA1CompleteExtend

Start of informative comment:

This capability terminates a pending SHA-1 calculation and EXTENDS the result into a Platform Configuration Register using a SHA-1 hash process.

This command is designed to complete a hash sequence and extend a PCR in memory-less environments.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SHA1CompleteExtend
4	4	2S	4	TPM_PCRINDEX	pcrNum	Index of the PCR to be modified
5	4	3S	4	UINT32	hashDataSize	Number of bytes in hashData, MUST be 64 or less
6	<>	4S	<>	BYTE [ ]	hashData	Final bytes to be hashed

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_SHA1CompleteExtend
4	20	3S	20	TPM_DIGEST	hashValue	The output of the SHA-1 hash.
5	20	4S	20	TPM_PCRVALUE	outDigest	The PCR value after execution of the command.

Description

This command SHALL incorporate a partial or complete block of data into the digest of an existing SHA-1 thread, EXTEND the resultant digest into a PCR, and terminate the SHA-1 session. hashDataSize MAY have values in the range of 0 through 64, inclusive.

The SHA-1 session MUST terminate even if the command returns an error, e.g. TPM_BAD_LOCALITY.

Actions

1. Map V1 to TPM_STANY_DATA
2. Map L1 to V1 -> localityModifier
3. If the current locality, held in L1, is not selected in TPM_PERMANENT_DATA -> pcrAttrib [pcrNum]. pcrExtendLocal, return TPM_BAD_LOCALITY

4. Create H1 the TPM_DIGEST of the SHA-1 session ensuring that hashData, if any, is added to the SHA-1 session
5. Perform the actions of TPM_Extend using H1 as the data and pcrNum as the PCR to extend

13.5 TPM_Sign

Start of informative comment:

The Sign command signs data and returns the resulting digital signature

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Sign.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
5	4	2s	4	UINT32	areaToSignSize	The size of the areaToSign parameter
6	<>	3s	<>	BYTE[]	areaToSign	The value to sign
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the use of keyHandle. HMAC key: key.usageAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Sign.
4	4	3S	4	UINT32	sigSize	The length of the returned digital signature
5	<>	4S	<>	BYTE[ ]	sig	The resulting digital signature.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

Description

The TPM MUST support all values of areaToSignSize that are legal for the defined signature scheme and key size. The maximum value of areaToSignSize is determined by the defined signature scheme and key size.

In the case of PKCS1v15_SHA1 the areaToSignSize MUST be TPM_DIGEST (the hash size of a SHA-1 operation - see 8.5.1 TPM_SS_RSASSAPKCS1v15_SHA1). In the case of PKCS1v15_DER the maximum size of areaToSign is k-11 octets, where k is limited by the key size (see TPM_SS_RSASSAPKCS1v15_DER).

Actions

1. The TPM validates the AuthData to use the key pointed to by keyHandle.
2. If the areaToSignSize is 0 the TPM returns TPM_BAD_PARAMETER.
3. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING or TPM_KEY_LEGACY, if not return the error code TPM_INVALID_KEYUSAGE
4. The TPM verifies that the signature scheme and key size can properly sign the areaToSign parameter.
5. If signature scheme is TPM_SS_RSASSAPKCS1v15_SHA1 then
    a. Validate that areaToSignSize is 20 return TPM_BAD_PARAMETER on error
    b. Set S1 to areaToSign
6. Else if signature scheme is TPM_SS_RSASSAPKCS1v15_DER then
    a. Validate that areaToSignSize is at least 11 bytes less than the key size, return TPM_BAD_PARAMETER on error
    b. Set S1 to areaToSign
7. else if signature scheme is TPM_SS_RSASSAPKCS1v15_INFO then
    a. Create S2 a TPM_SIGN_INFO structure
    b. Set S2 -> fixed to "SIGN"
    c. Set S2 -> replay to nonceOdd
        i. If nonceOdd is not present due to an unauthorized command return TPM_BAD_PARAMETER
    d. Set S2 -> dataLen to areaToSignSize
    e. Set S2 -> data to areaToSign
    f. Set S1 to the SHA-1(S2)
8. Else return TPM_INVALID_KEYUSAGE
9. The TPM computes the signature, sig, using the key referenced by keyHandle using S1 as the value to sign
10. Return the computed signature in Sig

13.6 TPM_GetRandom

Start of informative comment:

TPM_GetRandom returns the next bytesRequested bytes from the random number generator to the caller.

It is recommended that a TPM implement the RNG in a manner that would allow it to return RNG bytes such that the frequency of bytesRequested being more than the number of bytes available is an infrequent occurrence.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetRandom.
4	4	2S	4	UINT32	bytesRequested	Number of bytes to return

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetRandom.
4	4	3S	4	UINT32	randomBytesSize	The number of bytes returned
5	<>	4S	<>	BYTE[ ]	randomBytes	The returned bytes

Actions

1. The TPM determines if amount bytesRequested is available from the TPM.
2. Set randomBytesSize to the number of bytes available from the RNG. This number MAY be less than bytesRequested
3. Set randomBytes to the next randomBytesSize bytes from the RNG

13.7 TPM_StirRandom

Start of informative comment:

TPM_StirRandom adds entropy to the RNG state.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_StirRandom
4	4	2S	4	UINT32	dataSize	Number of bytes of input (<256)
5	<>	3S	<>	BYTE[ ]	inData	Data to add entropy to RNG state

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_StirRandom

Actions

The TPM updates the state of the current RNG using the appropriate mixing function.

13.8 TPM_CertifyKey

Start of informative comment:

The TPM_CertifyKey operation allows one key to certify the public portion of another key.

A TPM identity key may be used to certify non-migratable keys but is not permitted to certify migratory keys or certified migration keys. As such, it allows the TPM to make the statement "this key is held in a TPM-shielded location, and it will never be revealed." For this statement to have veracity, the Challenger must trust the policies used by the entity that issued the identity and the maintenance policy of the TPM manufacturer.

Signing and legacy keys may be used to certify both migratable and non-migratable keys. Then the usefulness of a certificate depends on the trust in the certifying key by the recipient of the certificate.

The key to be certified must be loaded before TPM_CertifyKey is called.

The determination to use the TPM_CERTIFY_INFO or TPM_CERTIFY_INFO2 on the output is based on which PCRs and what localities the certified key is restricted to. A key to be certified that does not have locality restrictions and which uses no PCRs greater than PCR #15 will cause this command return and sign a TPM_CERTIFY_INFO structure, which provides compatibility with V1.1 TPMs.

When this command is run to certify all other keys (those that use PCR #16 or higher, as well as those limited by locality in any way) it will return and sign a TPM_CERTIFY_INFO2 structure.

TPM_CertifyKey does not support the case where (a) the certifying key requires a usage authorization to be provided but (b) the key-to-be-certified does not. In such cases, TPM_CertifyKey2 must be used.

If a command tag (in the parameter array) specifies only one authorisation session, then the TPM convention is that the first session listed is ignored (authDataUsage must be NEVER for this key) and the incoming session data is used for the second auth session in the list. In TPM_CertifyKey, the first session is the certifying key and the second session is the key-to-be-certified. In TPM_CertifyKey2, the first session is the key-to-be-certified and the second session is the certifying key.

End of informative comment.

246 **Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CertifyKey
4	4			TPM_KEY_HANDLE	certHandle	Handle of the key to be used to certify the key.
5	4			TPM_KEY_HANDLE	keyHandle	Handle of the key to be certified.
6	20	2S	20	TPM_NONCE	antiReplay	160 bits of externally supplied data (typically a nonce provided to prevent replay -attacks)
7	4			TPM_AUTHHANDLE	certAuthHandle	The authorization session handle used for certHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	certAuth	The authorization session digest for inputs and certHandle. HMAC key: certKey.auth.
11	4			TPM_AUTHHANDLE	keyAuthHandle	The authorization session handle used for the key to be signed.
		2H2	20	TPM_NONCE	keylastNonceEven	Even nonce previously generated by TPM
12	20	3H2	20	TPM_NONCE	keynonceOdd	Nonce generated by system associated with keyAuthHandle
13	1	4H2	1	BOOL	continueKeySession	The continue use flag for the authorization session handle
14	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: key.usageAuth.

Outgoing Operands and Sizes

Param		HMAC		Type	Name	Description
#	Sz	#	Sz			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_CertifyKey
4	<>	3S	<>	TPM_CERTIFY_INFO	certifyInfo	TPM_CERTIFY_INFO or TPM_CERTIFY_INFO2 structure that provides information relative to keyhandle
5	4	4S	4	UINT32	outDataSize	The used size of the output area for outData
6	<>	5S	<>	BYTE[ ]	outData	The signature of certifyInfo
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag for cert key session
9	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters and parentHandle. HMAC key: certKey -> auth.
10	20	2H2	20	TPM_NONCE	keyNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	keynonceOdd	Nonce generated by system associated with keyAuthHandle
11	1	4H2	1	BOOL	continueKey Session	Continue use flag for target key session
12	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the target key. HMAC key: key.auth.

Actions

1. The TPM validates that the key pointed to by certHandle has a signature scheme of TPM_SS_RSASSAPKCS1v15_SHA1
2. Verify command and key AuthData values:
   a. If tag is TPM_TAG_RQU_AUTH2_COMMAND
      i. The TPM verifies the AuthData in certAuthHandle provides authorization to use the key pointed to by certHandle, return TPM_AUTHFAIL on error
      ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to use the key pointed to by keyHandle, return TPM_AUTH2FAIL on error
   b. else if tag is TPM_TAG_RQU_AUTH1_COMMAND
      i. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by certHandle, return TPM_AUTHFAIL on error.
      ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to use the key pointed to by keyHandle, return TPM_AUTHFAIL on error
   c. else if tag is TPM_TAG_RQU_COMMAND
      i. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by certHandle, return TPM_AUTHFAIL on error.

ii. Verify that authDataUsage is TPM_AUTH_NEVER or TPM_AUTH_PRIV_USE_ONLY for the key referenced by keyHandle, return TPM_AUTHFAIL on error.

3. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is TPM_KEY_IDENTITY)

a. If keyHandle -> keyFlags -> migratable is TRUE return TPM_MIGRATEFAIL

4. If keyHandle -> digestAtRelease requires the use of PCRs 16 or higher to calculate or if keyHandle -> localityAtRelease is not 0x1F

a. Set V1 to 1.2

5. Else

a. Set V1 to 1.1

6. If keyHandle -> pcrInfoSize is not 0

a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

i. Create a digestAtRelease according to the specified TPM_STCLEAR_DATA -> PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch return TPM_WRONGPCRVAL

ii. If specified validate any locality requests on error TPM_BAD_LOCALITY

b. If V1 is 1.1

i. Create C1 a TPM_CERTIFY_INFO structure

ii. Fill in C1 with the information from the key pointed to by keyHandle

iii. The TPM MUST set c1 -> pcrInfoSize to 44.

iv. The TPM MUST set c1 -> pcrInfo to a TPM_PCR_INFO structure properly filled out using the information from keyHandle.

v. The TPM MUST set c1 -> digestAtCreation to 20 bytes of 0x00.

c. Else

i. Create C1 a TPM_CERTIFY_INFO2 structure

ii. Fill in C1 with the information from the key pointed to by keyHandle

iii. Set C1 -> pcrInfoSize to the size of an appropriate TPM_PCR_INFO_SHORT structure.

iv. Set C1 -> pcrInfo to a properly filled out TPM_PCR_INFO_SHORT structure, using the information from keyHandle.

v. Set C1 -> migrationAuthoritySize to 0

7. Else

a. Create C1 a TPM_CERTIFY_INFO structure

b. Fill in C1 with the information from the key pointed to by keyHandle

c. The TPM MUST set c1 -> pcrInfoSize to 0

8. Create TPM_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note that <key> is the actual public modulus, and does not include any structure formatting.

9. Set C1 -> pubKeyDigest to H1
10. The TPM copies the antiReplay parameter to c1 -> data.
11. The TPM sets certifyInfo to C1.
12. The TPM creates m1, a message digest formed by taking the SHA1 of c1.
    a. The TPM then computes a signature using certHandle -> sigScheme. The resulting signed blob is returned in outData.

13.9 TPM_CertifyKey2

Start of informative comment:

This command is based on TPM_CertifyKey, but includes the ability to certify a Certifiable Migration Key (CMK), which requires extra input parameters.

TPM_CertifyKey2 always produces a TPM_CERTIFY_INFO2 structure.

TPM_CertifyKey2 does not support the case where (a) the key-to-be-certified requires a usage authorization to be provided but (b) the certifying key does not.

If a command tag (in the parameter array) specifies only one authorisation session, then the TPM convention is that the first session listed is ignored (authDataUsage must be NEVER for this key) and the incoming session data is used for the second auth session in the list. In TPM_CertifyKey2, the first session is the key to be certified and the second session is the certifying key.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_CertifyKey2
4	4			TPM_KEY_HANDLE	keyHandle	Handle of the key to be certified.
5	4			TPM_KEY_HANDLE	certHandle	Handle of the key to be used to certify the key.
6	20	2S	20	TPM_DIGEST	migrationPubDigest	The digest of a TPM_MSA_COMPOSITE structure, containing at least one public key of a Migration Authority
7	20	3S	20	TPM_NONCE	antiReplay	160 bits of externally supplied data (typically a nonce provided to prevent replay -attacks)
8	4			TPM_AUTHHANDLE	keyAuthHandle	The authorization session handle used for the key to be signed.
		2H1	20	TPM_NONCE	keylastNonceEven	Even nonce previously generated by TPM
9	20	3H1	20	TPM_NONCE	keynonceOdd	Nonce generated by system associated with keyAuthHandle
10	1	4H1	1	BOOL	continueKeySession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: key.usageAuth.
12	4			TPM_AUTHHANDLE	certAuthHandle	The authorization session handle used for certHandle.
		2H2	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
13	20	3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
14	1	4H2	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
15	20			TPM_AUTHDATA	certAuth	Authorization HMAC key: certKey.auth.

Outgoing Operands and Sizes

Param		HMAC		Type	Name	Description
#	Sz	#	Sz			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_CertifyKey2
4	<>	3S	<>	TPM_CERTIFY_INFO2	certifyInfo	TPM_CERTIFY_INFO2 relative to keyHandle
5	4	4S	4	UINT32	outDataSize	The used size of the output area for outData
6	<>	5S	<>	BYTE[ ]	outData	The signed public key.
7	20	2H1	20	TPM_NONCE	keyNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	keyNonceOdd	Nonce generated by system associated with certAuthHandle
8	1	4H1	1	BOOL	keyContinueAuthSession	Continue use flag for cert key session
9	20		20	TPM_AUTHDATA	keyResAuth	Authorization HMAC key: keyHandle -> auth.
10	20	2H2	20	TPM_NONCE	certNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	AuthLastNonceOdd	Nonce generated by system associated with certAuthHandle
11	1	4H2	1	BOOL	CertContinueAuthSession	Continue use flag for cert key session
12	20		20	TPM_AUTHDATA	certResAuth	Authorization HMAC key: certHandle -> auth.

Actions

1. The TPM validates that the key pointed to by certHandle has a signature scheme of TPM_SS_RSASSAPKCS1v15_SHA1
2. Verify command and key AuthData values:
   a. If tag is TPM_TAG_RQU_AUTH2_COMMAND
      i. The TPM verifies the AuthData in keyAuthHandle provides authorization to use the key pointed to by keyHandle, return TPM_AUTHFAIL on error
      ii. The TPM verifies the AuthData in certAuthHandle provides authorization to use the key pointed to by certHandle, return TPM_AUTH2FAIL on error
   b. else if tag is TPM_TAG_RQU_AUTH1_COMMAND
      i. Verify that authDataUsage is TPM_AUTH_NEVER or TPM_AUTH_PRIV_USE_ONLY for the key referenced by keyHandle, return TPM_AUTHFAIL on error
      ii. The TPM verifies the AuthData in certAuthHandle provides authorization to use the key pointed to by certHandle, return TPM_AUTH2FAIL on error
   c. else if tag is TPM_TAG_RQU_COMMAND
      i. Verify that authDataUsage is TPM_AUTH_NEVER or TPM_AUTH_PRIV_USE_ONLY for the key referenced by keyHandle, return TPM_AUTHFAIL on error
      ii. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by certHandle, return TPM_AUTHFAIL on error.

3. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is TPM_KEY_IDENTITY)
    - a. If keyHandle -> keyFlags -> migratable is TRUE and [keyHandle -> keyFlags-> migrateAuthority is FALSE or (keyHandle -> payload != TPM_PT_MIGRATE_RESTRICTED and keyHandle -> payload != TPM_PT_MIGRATE_EXTERNAL)] return TPM_MIGRATEFAIL
4. The TPM SHALL create a c1 a TPM_CERTIFY_INFO2 structure from the key pointed to by keyHandle
5. Create TPM_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note that <key> is the actual public modulus, and does not include any structure formatting.
6. Set C1 -> pubKeyDigest to H1
7. Copy the antiReplay parameter to c1 -> data
8. Copy other keyHandle parameters into C1
9. If keyHandle -> payload == TPM_PT_MIGRATE_RESTRICTED or TPM_PT_MIGRATE_EXTERNAL
    - a. create thisPubKey, a TPM_PUBKEY structure containing the public key, algorithm and parameters corresponding to keyHandle
    - b. Verify that the migration authorization is valid for this key
        - i. Create M2 a TPM_CMK_MIGAUTH structure
        - ii. Set M2 -> msaDigest to migrationPubDigest
        - iii. Set M2 -> pubkeyDigest to SHA1[thisPubKey]
        - iv. Verify that [keyHandle -> migrationAuth] == HMAC(M2) signed by using tpmProof as the secret and return error TPM_MA_SOURCE on mismatch
    - c. Set C1 -> migrationAuthority = SHA-1(migrationPubDigest || keyHandle -> payload)
    - d. if keyHandle -> payload == TPM_PT_MIGRATE_RESTRICTED
        - i. Set C1 -> payloadType = TPM_PT_MIGRATE_RESTRICTED
    - e. if keyHandle -> payload == TPM_PT_MIGRATE_EXTERNAL
        - i. Set C1 -> payloadType = TPM_PT_MIGRATE_EXTERNAL
10. Else
    - a. set C1 -> migrationAuthority = NULL
    - b. set C1 -> migrationAuthoritySize =0
    - c. Set C1 -> payloadType = TPM_PT_ASYM
11. If keyHandle -> pcrInfoSize is not 0
    - a. The TPM MUST set c1 -> pcrInfoSize to match the pcrInfoSize from the keyHandle key.
    - b. The TPM MUST set c1 -> pcrInfo to match the pcrInfo from the keyHandle key
    - c. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

    i. Create a digestAtRelease according to the specified TPM_STCLEAR_DATA -> PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch return TPM_WRONGPCRVAL

    ii. If specified validate any locality requests on error TPM_BAD_LOCALITY

12. Else

    a. The TPM MUST set c1 -> pcrInfoSize to 0

13. The TPM creates m1, a message digest formed by taking the SHA1 of c1

    a. The TPM then computes a signature using certHandle -> sigScheme. The resulting signed blob is returned in outData

14. Endorsement Key Handling

Start of informative comment:

There are two create EK commands. The first matches the 1.1 functionality. The second provides the mechanism to enable revokeEK.

The TPM and platform manufacturer decide on the inclusion or exclusion of the ability to execute revokeEK.

The restriction to have the TPM generate the EK does not remove the manufacturing option to "squirt" the EK. During manufacturing, the TPM does not enforce all protections or requirements; hence, the restriction on only TPM generation of the EK is also not in force.

End of informative comment.

1. A TPM SHALL NOT install an EK unless generated on the TPM by execution of TPM_CreateEndorsementKeyPair or TPM_CreateRevocableEK

14.1 TPM_CreateEndorsementKeyPair

Start of informative comment:

This command creates the TPM endorsement key. It returns a failure code if an endorsement key already exists.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM _ORD_CreateEndorsementKeyPair
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data
5	<>	3S	<>	TPM_KEY_PARMS	keyInfo	Information about key to be created, this includes all algorithm parameters

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateEndorsementKeyPair
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay

Actions

1. If an EK already exists, return TPM_DISABLED_CMD
2. Validate the keyInfo parameters for the key description
    a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For interoperability the key length SHOULD be 2048
    b. If the algorithm type is other than RSA the strength provided by the key MUST be comparable to RSA 2048
    c. The other parameters of keyInfo (signatureScheme etc.) are ignored.
3. Create a key pair called the "endorsement key pair" using a TPM-protected capability. The type and size of key are that indicated by keyInfo
4. Create checksum by performing SHA1 on the concatenation of (PUBEK || antiReplay)
5. Store the PRIVEK
6. Create TPM_PERMANENT_DATA -> tpmDAASeed from the TPM RNG

7. Set TPM_PERMANENT_FLAGS -> CEKPUsed to TRUE
8. Set TPM_PERMANENT_FLAGS -> enableRevokeEK to FALSE

14.2 TPM_CreateRevocableEK

Start of informative comment:

This command creates the TPM endorsement key. It returns a failure code if an endorsement key already exists. The TPM vendor may have a separate mechanism to create the EK and "squirt" the value into the TPM.

The input parameters specify whether the EK is capable of being reset, whether the AuthData value to reset the EK will be generated by the TPM, and the new AuthData value itself if it is not to be generated by the TPM. The output parameter is the new AuthData value that must be used when resetting the EK (if it is capable of being reset).

The command TPM_RevokeTrust must be used to reset an EK (if it is capable of being reset).

Owner authorisation is unsuitable for authorizing resetting of an EK: someone with Physical Presence can remove a genuine Owner, install a new Owner, and revoke the EK. The genuine Owner can reinstall, but the platform will have lost its original attestation and may not be trusted by challengers. Therefore if a password is to be used to revoke an EK, it must be a separate password, given to the genuine Owner.

In v1.2 an OEM has extra choices when creating EKs.

a) An OEM could manufacture all of its TPMs with enableRevokeEK==TRUE.

If the OEM has tracked the EKreset passwords for these TPMs, the OEM can give the passwords to customers. The customers can use the passwords as supplied, change the passwords, or clear the EKs and create new EKs with new passwords.

If EKreset passwords are random values, the OEM can discard those values and not give them to customers. There is then a low probability (statistically zero) chance of a local DOS attack to reset the EK by guessing the password. The chance of a remote DOS attack is zero because Physical Presence must also be asserted to use TPM_RevokeTrust.

b) An OEM could manufacture some of its TPMs with enableRevokeEK==FALSE. Then the EK can never be revoked, and the chance of even a local DOS attack on the EK is eliminated.

End of informative comment.

This is an optional command

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateRevocableEK
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data
5	<>	3S	<>	TPM_KEY_PARMS	keyInfo	Information about key to be created, this includes all algorithm parameters
6	1	4S	1	BOOL	generateReset	If TRUE use TPM RNG to generate EKreset. If FALSE use the passed value inputEKreset
7	20	5S	20	TPM_NONCE	inputEKreset	The authorization value to be used with TPM_RevokeTrust if generateReset==FALSE, else the parameter is present but ignored

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateRevocableEK
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay
6	20	5S	20	TPM_NONCE	outputEKreset	The AuthData value to use TPM_RevokeTrust

Actions

1. If an EK already exists, return TPM_DISABLED_CMD
2. Perform the actions of TPM_CreateEndorsementKeyPair, if any errors return with error
3. Set TPM_PERMANENT_FLAGS -> enableRevokeEK to TRUE
    a. If generateReset is TRUE then
        i. Set TPM_PERMANENT_DATA -> EKreset to the next value from the TPM RNG
    b. Else
        i. Set TPM_PERMANENT_DATA -> EKreset to inputEKreset
4. Return PUBEK, checksum and Ekreset
5. Create TPM_PERMANENT_DATA -> tpmDAASeed from the TPM RNG
6. The outputEKreset AuthData is sent in the clear. There is no uniqueness on the TPM available to actually perform encryption or use an encrypted channel. The assumption is that this operation is occurring in a controlled environment and sending the value in the clear is acceptable.

14.3 TPM_RevokeTrust

Start of informative comment:

This command clears the EK and sets the TPM back to a pure default state. The generation of the AuthData value occurs during the generation of the EK. It is the responsibility of the EK generator to properly protect and disseminate the RevokeTrust AuthData.

End of informative comment.

This is an optional command

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_RevokeTrust
4	20	2S	20	TPM_DIGEST	EKReset	The value that will be matched to EK Reset

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_RevokeTrust

Actions

1. The TPM MUST validate that TPM_PERMANENT_FLAGS -> enableRevokeEK is TRUE, return TPM_PERMANENTEK on error
2. The TPM MUST validate that the EKReset matches TPM_PERMANENT_DATA -> EKReset return TPM_AUTHFAIL on error.
3. Ensure that physical presence is being asserted
4. Perform the actions of TPM_OwnerClear (excepting the command authentication)
    a. NV items with the pubInfo -> nvIndex D value set MUST be deleted. This changes the TPM_OwnerClear handling of the same NV areas

    b. Set TPM_PERMANENT_FLAGS -> nvLocked to FALSE
5. Invalidate TPM_PERMANENT_DATA -> tpmDAASeed
6. Invalidate the EK and any internal state associated with the EK

14.4 TPM_ReadPubek

Start of informative comment:

Return the endorsement key public portion. This value should have controls placed upon access, as it is a privacy sensitive value.

The readPubek flag is set to FALSE by TPM_TakeOwnership and set to TRUE by TPM_OwnerClear, thus mirroring if a TPM Owner is present.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadPubek
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadPubek
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay

Description

This command returns the PUBEK.

Actions

The TPM_ReadPubek command SHALL

1. If TPM_PERMANENT_FLAGS -> readPubek is FALSE return TPM_DISABLED_CMD
2. If no EK is present the TPM MUST return TPM_NO_ENDORSEMENT
3. Create checksum by performing SHA1 on the concatenation of (pubEndorsementKey || antiReplay).
4. Export the PUBEK and checksum.

14.5 TPM_OwnerReadInternalPub

Start of informative comment:

A TPM Owner authorized command that returns the public portion of the EK or SRK.

The keyHandle parameter is included in the incoming session authorization to prevent alteration of the value, causing a different key to be read. Unlike most key handles, which can be mapped by higher layer software, this key handle has only two fixed values.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerReadInternalPub
4	4	2S	4	TPM_KEY_HANDLE	keyHandle	Handle for either PUBEK or SRK
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for he authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerReadInternalPub
4	<>	3S	<>	TPM_PUBKEY	publicPortion	The public portion of the requested key
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Actions

1. Validate the parameters and TPM Owner AuthData for this command
2. If keyHandle is TPM_KH_EK
    a. Set publicPortion to PUBEK

3. Else If keyHandle is TPM_KH_SRK
    a. Set publicPortion to the TPM_PUBKEY of the SRK
4. Else return TPM_BAD_PARAMETER
5. Export the public key of the referenced key

15. Identity Creation and Activation

15.1 TPM_MakeIdentity

Start of informative comment:

Generate a new Attestation Identity Key (AIK)

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_MakeIdentity.
4	20	2S	20	TPM_ENCAUTH	identityAuth	Encrypted usage AuthData for the new identity
5	20	3S	20	TPM_CHOSENID_HASH	labelPrivCADigest	The digest of the identity label and privacy CA chosen for the AIK
6	<>	4S	<>	TPM_KEY	idKeyParams	Structure containing all parameters of new identity key. pubKey.keyLength & idKeyParams.encData are both 0MAY be TPM_KEY12
7	4			TPM_AUTHHANDLE	srkAuthHandle	The authorization session handle used for SRK authorization.
		2H1	20	TPM_NONCE	srkLastNonceEven	Even nonce previously generated by TPM
8	20	3H1	20	TPM_NONCE	srknonceOdd	Nonce generated by system associated with srkAuthHandle
9	1	4H1	1	BOOL	continueSrkSession	Ignored
10	20			TPM_AUTHDATA	srkAuth	The authorization session digest for the inputs and the SRK. HMAC key: srk.usageAuth.
11	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication. Session type MUST be OSAP.
		2H2	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
12	20	3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
13	1	4H2	1	BOOL	continueAuthSession	Ignored
14	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal:TPM_ORD_MakeIdentity.
4	<>	3S	<>	TPM_KEY	idKey	The newly created identity key . MAY be TPM_KEY12
5	4	4S	4	UINT32	identityBindingSize	The used size of the output area for identityBinding
6	<>	5S	<>	BYTE[ ]	identityBinding	Signature of TPM_IDENTITY_CONTENTS using idKey.private.
7	20	2H2	20	TPM_NONCE	srkNonceEven	Even nonce newly generated by TPM.
		3H2	20	TPM_NONCE	srknonceOdd	Nonce generated by system associated with srkAuthHandle
8	1	4H2	1	BOOL	continueSrkSession	Continue use flag. Fixed value of FALSE
9	20			TPM_AUTHDATA	srkAuth	The authorization session digest used for the outputs and srkAuth session. HMAC key: srk.usageAuth.
10	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	Continue use flag. Fixed value of FALSE
12	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Description

The public key of the new TPM identity SHALL be identityPubKey. The private key of the new TPM identity SHALL be tpm_signature_key.

Properties of the new identity

Type	Name	Description
TPM_PUBKEY	identityPubKey	This SHALL be the public key of a previously unused asymmetric key pair.
TPM_STORE_ASYMKEY	tpm_signature_key	This SHALL be the private key that forms a pair with identityPubKey and SHALL be extant only in a TPM-shielded location.

This capability also generates a TPM_KEY containing the tpm_signature_key.

If identityPubKey is stored on a platform it SHALL exist only in storage to which access is controlled and is available to authorized entities.

Actions

A Trusted Platform Module that receives a valid TPM_MakeIdentity command SHALL do the following:

1. Validate the idKeyParams parameters for the key description

   a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For interoperability the key length SHOULD be 2048

b. If the algorithm type is other than RSA the strength provided by the key MUST be comparable to RSA 2048

c. If the TPM is not designed to create a key of the requested type, return the error code TPM_BAD_KEY_PROPERTY

d. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then

i. If authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

2. Use authHandle to verify that the Owner authorized all TPM_MakeIdentity input parameters.

3. Use srkAuthHandle to verify that the SRK owner authorized all TPM_MakeIdentity input parameters.

4. Verify that idKeyParams -> keyUsage is TPM_KEY_IDENTITY. If it is not, return TPM_INVALID_KEYUSAGE

5. Verify that idKeyParams -> keyFlags -> migratable is FALSE. If it is not, return TPM_INVALID_KEYUSAGE

6. If authHandle indicates XOR encryption for the AuthData secrets

a. Create X1 the SHA-1 of the concatenation of (ownerAuth -> sharedSecret || authLastNonceEven)

b. Create a1 by XOR X1 and identityAuth

7. Else

a. Create a1 by decrypting identityAuth using the algorithm indicated in the OSAP session

b. Key is from ownerAuth -> sharedSecret

c. IV is SHA-1 of (authLastNonceEven || nonceOdd)

8. Set continueAuthSession and continueSRKSession to FALSE.

9. Determine the structure version

a. If idKeyParms -> tag is TPM_TAG_KEY12

i. Set V1 to 2

ii. Create idKey a TPM_KEY12 structure using idKeyParams as the default values for the structure

b. If idKeyParms -> ver is 1.1

i. Set V1 to 1

ii. Create idKey a TPM_KEY structure using idKeyParams as the default values for the structure

10. Set the digestAtCreation values for pcrInfo

a. For TPM_PCR_INFO_LONG include the locality of the current command

11. Create an asymmetric key pair (identityPubKey and tpm_signature_key) using a TPM-protected capability, in accordance with the algorithm specified in idKeyParams

12. Ensure that the AuthData information in A1 is properly stored in the idKey as usageAuth.
13. Attach identityPubKey and tpm_signature_key to idKey
14. Set idKey -> migrationAuth to TPM_PERMANENT_DATA-> tpmProof
15. Ensure that all TPM_PAYLOAD_TYPE structures identify this key as TPM_PT_ASYM
16. Encrypt the private portion of idKey using the SRK as the parent key
17. Create a TPM_IDENTITY_CONTENTS structure named idContents using labelPrivCADigest and the information from idKey
18. Sign idContents using tpm_signature_key and TPM_SS_RSASSAPKCS1v15_SHA1. Store the result in identityBinding.

15.2 TPM_ActivateIdentity

Start of informative comment:

The purpose of TPM_ActivateIdentity is to twofold. The first purpose is to obtain assurance that the credential in the TPM_SYM_CA_ATTESTATION is for this TPM. The second purpose is to obtain the session key used to encrypt the TPM_IDENTITY_CREDENTIAL.

This is an extension to the 1.1 functionality of TPM_ActivateIdentity. The blob sent to from the CA can be in the 1.1 format or the 1.2 format. The TPM determines the type from the size or version information in the blob.

TPM_ActivateIdentity checks that the symmetric session key corresponds to a TPM-identity before releasing that session key.

Only the Owner of the TPM has the privilege of activating a TPM identity. The Owner is required to authorize the TPM_ActivateIdentity command. The owner may authorize the command using either the TPM_OIAP or TPM_OSAP authorization protocols.

The creator of the ActivateIdentity package can specify if any PCR values are to be checked before releasing the session key.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ActivateIdentity
4	4			TPM_KEY_HANDLE	idKey Handle	Identity key to be activated
5	4	2S	4	UINT32	blobSize	Size of encrypted blob from CA
6	<>	3S	<>	BYTE [ ]	blob	The encrypted ASYM_CA_CONTENTS or TPM_EK_BLOB
7	4			TPM_AUTHHANDLE	idKeyAuthHandle	The authorization session handle used for ID key authorization.
		2H1	20	TPM_NONCE	idKeyLastNonceEven	Even nonce previously generated by TPM
8	20	3H1	20	TPM_NONCE	idKeynonceOdd	Nonce generated by system associated with idKeyAuthHandle
9	1	4H1	1	BOOL	continueIdKeySession	Continue usage flag for idKeyAuthHandle.
10	20			TPM_AUTHDATA	idKeyAuth	The authorization session digest for the inputs and ID key. HMAC key: idKey.usageAuth.
11	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H2	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
12	20	3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
13	1	4H2	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
14	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal:TPM_ORD_ActivateIdentity
4	<>	3S	<>	TPM_SYMMETRIC_KEY	symmetricKey	The decrypted symmetric key.
5	20	2H1	20	TPM_NONCE	idKeyNonceEven	Even nonce newly generated by TPM.
		3H1	20	TPM_NONCE	idKeynonceOdd	Nonce generated by system associated with idKeyAuthHandle
6	1	4H1	1	BOOL	continueIdKeySession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	idKeyAuth	The authorization session digest used for the returned parameters and idKeyAuth session. HMAC key: idKey.usageAuth.
8	20	2H2	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H2	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Description

1. The command TPM_ActivateIdentity activates a TPM identity created using the command TPM_MakeIdentity.
2. The command assumes the availability of the private key associated with the identity. The command will verify the association between the keys during the process.
3. The command will decrypt the input blob and extract the session key and verify the connection between the public and private keys. The input blob can be in 1.1 or 1.2 format.

Actions

A Trusted Platform Module that receives a valid TPM_ActivateIdentity command SHALL do the following:

1. Using the authHandle field, validate the owner's AuthData to execute the command and all of the incoming parameters.
2. Using the idKeyAuthHandle, validate the AuthData to execute command and all of the incoming parameters
3. Validate that the idKey is the public key of a valid TPM identity by checking that idKeyHandle -> keyUsage is TPM_KEY_IDENTITY. Return TPM_BAD_PARAMETER on mismatch
4. Create H1 the digest of a TPM_PUBKEY derived from idKey
5. Decrypt blob creating B1 using PRIVEK as the decryption key

```
6. Determine the type and version of B1
   a. If B1 -> tag is TPM_TAG_EK_BLOB then
      i. B1 is a TPM_EK_BLOB
   b. Else
      i. B1 is a TPM_ASYM_CA_CONTENTS. As there is no tag for this structure it is possible for the TPM to make a mistake here but other sections of the structure undergo validation
7. If B1 is a version 1.1 TPM_ASYM_CA_CONTENTS then
   a. Compare H1 to B1 -> idDigest on mismatch return TPM_BAD_PARAMETER
   b. Set K1 to B1 -> sessionKey
8. If B1 is a TPM_EK_BLOB then
   a. Validate that B1 -> ekType is TPM_EK_TYPE_ACTIVATE, return TPM_BAD_TYPE if not.
   b. Assign A1 as a TPM_EK_BLOB_ACTIVATE structure from B1 -> blob
   c. Compare H1 to A1 -> idDigest on mismatch return TPM_BAD_PARAMETER
   d. If A1 -> pcrSelection is not NULL
      i. Compute a composite hash C1 using the PCR selection A1 -> pcrSelection
      ii. Compare C1 to A1 -> pcrInfo->digestAtRelease and return TPM_WRONGPCRVAL on a mismatch
      iii. If A1 -> pcrInfo specifies a locality ensure that the appropriate locality has been asserted, return TPM_BAD_LOCALITY on error
   e. Set K1 to A1 -> symmetricKey
9. Return K1
```

16. Integrity Collection and Reporting

Start of informative comment:

This section deals with what commands have direct access to the PCR

End of informative comment.

1. The TPM SHALL only allow the following commands to alter the value of TPM_STCLEAR_DATA -> PCR
   a. TPM_Extend
   b. TPM_SHA1CompleteExtend
   c. TPM_Startup
   d. TPM_PCR_Reset

16.1 TPM_Extend

Start of informative comment:

This adds a new measurement to a PCR

End of informative comment:

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Extend.
4	4	2S	4	TPM_PCRINDEX	pcrNum	The PCR to be updated.
5	20	3S	20	TPM_DIGEST	inDigest	The 160 bit value representing the event to be recorded.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Extend.
4	20	3S	20	TPM_PCRVALUE	outDigest	The PCR value after execution of the command.

Descriptions

Add a measurement value to a PCR

Actions

1. Map L1 to TPM_STANY_FLAGS -> localityModifier
2. Map P1 to TPM_PERMANENT_DATA -> pcrAttrib [pcrNum]. pcrExtendLocal
3. If, for the value of L1, the corresponding bit is not set in the bit map P1, return TPM_BAD_LOCALITY
4. Create c1 by concatenating (TPM_STCLEAR_DATA -> PCR[pcrNum] || inDigest). This takes the current PCR value and concatenates the inDigest parameter.
5. Create h1 by performing a SHA1 digest of c1.
6. Store h1 to TPM_STCLEAR_DATA -> PCR[pcrNum]
7. If TPM_PERMANENT_FLAGS -> disable is TRUE or TPM_STCLEAR_FLAGS -> deactivated is TRUE
    a. Set outDigest to 20 bytes of 0x00

8. Else
    a. Set outDigest to h1

16.2 TPM_PCRRead

Start of informative comment:

The TPM_PCRRead operation provides non-cryptographic reporting of the contents of a named PCR.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_PCRRead
4	4	2S	4	TPM_PCRINDEX	pcrIndex	Index of the PCR to be read

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_PCRRead
4	20	3S	20	TPM_PCRVALUE	outDigest	The current contents of the named PCR

Description

The TPM_PCRRead operation returns the current contents of the named register to the caller.

Actions

1. Set outDigest to TPM_STCLEAR_DATA -> PCR[pcrIndex]
2. Return TPM_SUCCESS

16.3 TPM_Quote

Start of informative comment:

The TPM_Quote operation provides cryptographic reporting of PCR values. A loaded key is required for operation. TPM_Quote uses a key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

The term "ExternalData" is used because an important use of TPM_Quote is to provide a digital signature on arbitrary data, where the signature includes the PCR values of the platform at time of signing. Hence the "ExternalData" is not just for anti-replay purposes, although it is (of course) used for that purpose in an integrity challenge.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Quote.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can sign the PCR values.
5	20	2S	20	TPM_NONCE	externalData	160 bits of externally supplied data (typically a nonce provided by a server to prevent replay -attacks)
6	<>	3S	<>	TPM_PCR_SELECTION	targetPCR	The indices of the PCRs that are to be reported.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and keyHandle. HMAC key: key -> usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Quote.
4	<>	3S	<>	TPM_PCR_COMPOSITE	pcrData	A structure containing the same indices as targetPCR, plus the corresponding current PCR values.
5	4	4S	4	UINT32	sigSize	The used size of the output area for the signature
6	<>	5S	<>	BYTE[ ]	sig	The signed data blob.
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
9	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: Key -> usageAuth.

Actions

1. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.
2. The keyHandle -> sigScheme MUST use SHA-1, return TPM_INAPPROPRIATE_SIG if it does not
3. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY, or TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE
4. Validate targetPCR
    a. targetPCR is a valid TPM_PCR_SELECTION structure
    b. On errors return TPM_INVALID_PCR_INFO
5. Create H1 a SHA-1 hash of a TPM_PCR_COMPOSITE using the TPM_STCLEAR_DATA -> PCR indicated by targetPCR -> pcrSelect
6. Create Q1 a TPM_QUOTE_INFO structure
    a. Set Q1 -> version to 1.1.0.0
    b. Set Q1 -> fixed to “QUOT”
    c. Set Q1 -> digestValue to H1
    d. Set Q1 -> externalData to externalData
7. Sign SHA-1 hash of Q1 using keyHandle as the signature key
8. Return the signature in sig

16.4 TPM_PCR_Reset

Start of informative comment:

For PCR with the pcrReset attribute set to TRUE, this command resets the PCR back to the default value, this mimics the actions of TPM_Init. The PCR may have restrictions as to which locality can perform the reset operation.

Sending a null pcrSelection results in an error is due to the requirement that the command actually do something. If pcrSelection is null there are no PCR to reset and the command would then do nothing.

For PCR that are resettable, the presence of a Trusted Operating System (TOS) can change the behavior of TPM_PCR_Reset. The following pseudo code shows how the behavior changes

```
At TPM_Startup
    If TPM_PCR_ATTRIBUTES->pcrReset is FALSE
        Set PCR to 0x00...00
    Else
        Set PCR to 0xFF...FF
At TPM_PCR_Reset
    If TPM_PCR_ATTRIBUTES->pcrReset is TRUE
        If TOSPresent
                Set PCR to 0x00...00
        Else
                Set PCR to 0xFF...FF
    Else
        Return error
```

The above pseudocode is for example only, for the details of a specific platform, the reader must review the platform specific specification. The purpose of the above pseudocode is to show that both pcrReset and the TOSPresent bit control the value in use to when the PCR resets.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCR_Reset
4	<>	2S	<>	TPM_PCR_SELECTION	pcrSelection	The PCR's to reset

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCR_Reset

Descriptions

This command resets PCR values back to the default value. The command MUST validate that all PCR registers that are selected are available to be reset before resetting any PCR. This command MUST either reset all selected PCR registers or none of the PCR registers.

Actions

1. Validate that pcrSelection is valid
   a. is a valid TPM_PCR_SELECTION structure
   b. pcrSelection -> pcrSelect is non-zero
   c. On errors return TPM_INVALID_PCR_INFO
2. Map L1 to TPM_STANY_FLAGS -> localityModifier
3. For each PCR selected perform the following
   a. If TPM_PERMANENT_DATA -> pcrAttrib[pcrIndex].pcrReset is FALSE, return TPM_NOTRESETABLE
   b. If, for the value L1, the corresponding bit is clear in the bit map TPM_PERMANENT_DATA -> pcrAttrib[pcrIndex].pcrResetLocal, return TPM_NOTLOCAL
4. For each PCR selected perform the following
   a. The PCR MAY only reset to 0x00...00 or 0xFF...FF
   b. The logic to determine which value to use MUST be described by a platform specific specification

16.5 TPM_Quote2

Start of informative comment:

The TPM_Quote2 operation provides cryptographic reporting of PCR values. A loaded key is required for operation. TPM_Quote2 uses a key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

The term "externalData" is used because an important use of TPM_Quote2 is to provide a digital signature on arbitrary data, where the signature includes the PCR values of the platform at time of signing. Hence the "externalData" is not just for anti-replay purposes, although it is (of course) used for that purpose in an integrity challenge.

TPM_Quote2 differs from TPM_Quote in that TPM_Quote2 uses TPM_PCR_INFO_SHORT to hold information relative to the PCR registers. TPM_PCR_INFO_SHORT includes locality information to provide the requestor a more complete view of the current platform configuration.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Quote2
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can sign the PCR values.
5	20	2S	20	TPM_NONCE	externalData	160 bits of externally supplied data (typically a nonce provided by a server to prevent replay -attacks)
6	<>	3S	<>	TPM_PCR_SELECTION	targetPCR	The indices of the PCRs that are to be reported.
7	1	4S	1	BOOL	addVersion	When TRUE add TPM_CAP_VERSION_INFO to the output
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and keyHandle. HMAC key: key -> usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Quote2
4	<>	3S	<>	TPM_PCR_INFO_SHORT	pcrData	The value created and signed for the quote
5	4	4S	4	UINT32	versionInfoSize	Size of the version info
6	<>	5S	<>	TPM_CAP_VERSION_INFO	versionInfo	The version info
7	4	6S	4	UINT32	sigSize	The used size of the output area for the signature
8	<>	7S	<>	BYTE[ ]	sig	The signed data blob.
9	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
11	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: Key -> usageAuth.

Actions

1. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.
2. The keyHandle -> sigScheme MUST use SHA-1, return TPM_INAPPROPRIATE_SIG if it does not
3. Validate targetPCR is a valid TPM_PCR_SELECTION structure, on errors return TPM_INVALID_PCR_INFO
4. Create H1 a SHA-1 hash of a TPM_PCR_COMPOSITE using the TPM_STCLEAR_DATA -> PCR[] indicated by targetPCR -> pcrSelect
5. Create S1 a TPM_PCR_INFO_SHORT
    a. Set S1->pcrSelection to targetPCR
    b. Set S1->localityAtRelease to TPM_STANY_DATA -> localityModifier
    c. Set S1->digestAtRelease to H1
6. Create Q1 a TPM_QUOTE_INFO2 structure
    a. Set Q1 -> fixed to "QUT2"
    b. Set Q1 -> infoShort to S1
    c. Set Q1 -> externalData to externalData
7. If addVersion is TRUE
    a. Concatenate to Q1 a TPM_CAP_VERSION_INFO structure
    b. Set the output parameters for versionInfo

8. Else
   a. Set versionInfoSize to 0
   b. Return no bytes in versionInfo
9. Sign a SHA-1 hash of Q1 using keyHandle as the signature key
10. Return the signature in sig

17. Changing AuthData

17.1 TPM_ChangeAuth

Start of informative comment:

The TPM_ChangeAuth command allows the owner of an entity to change the AuthData for the entity.

TPM_ChangeAuth requires the encryption of one parameter ("NewAuth"). For the sake of uniformity with other commands that require the encryption of more than one parameter, the parameters used for used encryption are generated from the authLastNonceEven (created during the OSAP session), nonceOdd, and the session shared secret.

The parameter list to this command must always include two authorization sessions, regardless of the state of authDataUsage for the respective keys.

End of informative comment.

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuth
4	4			TPM_KEY_HANDLE	parentHandle	Handle of the parent key to the entity.
5	2	2 S	2	TPM_PROTOCOL_ID	protocolID	The protocol in use.
6	20	3 S	20	TPM_ENCAUTH	newAuth	The encrypted new AuthData for the entity. The encryption key is the shared secret from the OSAP protocol.
7	2	4 S	2	TPM_ENTITY_TYPE	entityType	The type of entity to be modified
8	4	5 S	4	UINT32	encDataSize	The size of the encData parameter
9	<>	6 S	<>	BYTE[ ]	encData	The encrypted entity that is to be modified.
10	4			TPM_AUTHHANDLE	parentAuthHandle	The authorization session handle used for the parent key.
		2 H1	20	TPM_NONCE	authLastNonceEven	Even nonc e previously generated by TPM to cover inputs
11	20	3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
12	1	4 H1	1	BOOL	continueAuthSession	Ignored, parentAuthHandle is always terminated.
13	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.
14	4			TPM_AUTHHANDLE	entityAuthHandle	The authorization session handle used for the encrypted entity. The session type MUST be OIAP
		2 H2	20	TPM_NONCE	entitylastNonceEven	Even nonce previously generated by TPM
15	20	3 H2	20	TPM_NONCE	entitynonceOdd	Nonce generated by system associated with entityAuthHandle
16	1	4 H2	1	BOOL	continueEntitySession	Ignored, entityAuthHandle is always terminated.
17	20			TPM_AUTHDATA	entityAuth	The authorization session digest for the inputs and encrypted entity. HMAC key: entity.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation. See section 4.3.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_ChangeAuth
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[]	outData	The modified, encrypted entity.
6	20	2 H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
7	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, fixed value of FALSE
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters and parentHandle. HMAC key: parentKey.usageAuth.
9	20	2 H2	20	TPM_NONCE	entityNonceEven	Even nonce newly generated by TPM to cover entity
		3 H2	20	TPM_NONCE	entitynonceOdd	Nonce generated by system associated with entityAuthHandle
10	1	4 H2	1	BOOL	continueEntity Session	Continue use flag, fixed value of FALSE
11	20			TPM_AUTHDATA	entityAuth	The authorization session digest for the returned parameters and entity. HMAC key: entity.usageAuth, the original and not the new auth value

Description

1. The parentAuthHandle session type MUST be TPM_PID_OSAP.
2. In this capability, the SRK cannot be accessed as entityType TPM_ET_KEY, since the SRK is not wrapped by a parent key.

Actions

1. Verify that entityType is one of TPM_ET_DATA, TPM_ET_KEY and return the error TPM_WRONG_ENTITYTYPE if not.
2. Verify that parentAuthHandle session type is TPM_PID_OSAP return TPM_BAD_MODE on error
3. Verify that entityAuthHandle session type is TPM_PID_OIAP return TPM_BAD_MODE on error
4. The encData parameter MUST be the encData field from either the TPM_STORED_DATA or TPM_KEY structures.
5. If parentAuthHandle indicates XOR encryption for the AuthData secrets
    a. Create X1 the SHA-1 of the concatenation of (parentAuthHandle -> sharedSecret || authLastNonceEven)
    b. Create decryptAuth by XOR X1 and newAuth
6. Else

a. Create newAuth by decrypting newAuth using the algorithm indicated in the OSAP session

b. Key is from parentAuthHandle -> sharedSecret

c. IV is SHA-1 of (authLastNonceEven | | nonceOdd)

7. The TPM MUST validate the command using the AuthData in the parentAuth parameter

8. After parameter validation the TPM creates b1 by decrypting encData using the key pointed to by parentHandle.

9. The TPM MUST validate that b1 is a valid TPM structure, either a TPM_STORE_ASYMKEY or a TPM_SEALED_DATA

a. Check the tag, length and authValue for match, return TPM_INVALID_STRUCTURE on any mismatch

10. The TPM replaces the AuthData for b1 with decryptAuth created above.

11. The TPM encrypts b1 using the appropriate mechanism for the type using the parentKeyHandle to provide the key information.

12. The TPM MUST enforce the destruction of both the parentAuthHandle and entityAuthHandle sessions.

17.2 TPM_ChangeAuthOwner

Start of informative comment:

The TPM_ChangeAuthOwner command allows the owner of an entity to change the AuthData for the TPM Owner or the SRK.

This command requires authorization from the current TPM Owner to execute.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthOwner
4	2	2S	2	TPM_PROTOCOL_ID	protocolID	The protocol in use.
5	20	3S	20	TPM_ENCAUTH	newAuth	The encrypted new AuthData for the entity. The encryption key is the shared secret from the OSAP protocol.
6	2	4S	2	TPM_ENTITY_TYPE	entityType	The type of entity to be modified
7	4			TPM_AUTHHANDLE	ownerAuthHandle	The authorization session handle used for the TPM Owner.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with ownerAuthHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag the TPM ignores this value
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and ownerHandle. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_ChangeAuthOwner
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with ownerAuthHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed value of FALSE
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters and ownerHandle. HMAC key: ownerAuth, the original value and not the new auth value

Actions

1. The TPM MUST validate the command using the AuthData in the ownerAuth parameter
2. The ownerAuthHandle session type MUST be TPM_PID_OSAP
3. Verify that entityType is either TPM_ET_OWNER or TPM_ET_SRK, and return the error TPM_WRONG_ENTITYTYPE if not.
4. If ownerAuthHandle indicates XOR encryption for the AuthData secrets
    a. Create X1 the SHA-1 of the concatenation of (ownerAuthHandle -> sharedSecret || authLastNonceEven)
    b. Create decryptAuth by XOR X1 and newAuth
5. Else
    a. Create newAuth by decrypting newAuth using the algorithm indicated in the OSAP session
    b. Key is the previous ownerAuth
    c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
6. The TPM MUST enforce the destruction of the ownerAuthHandle session upon completion of this command (successful or unsuccessful). This includes setting continueAuthSession to FALSE
7. Set the AuthData for the indicated entity to decryptAuth
8. Invalidate all sessions, active or saved

18. Authorization Sessions

18.1 TPM_OIAP

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OIAP.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OIAP.
4	4			TPM_AUTHHANDLE	authHandle	Handle that TPM creates that points to the authorization state.
5	20			TPM_NONCE	nonceEven	Nonce generated by TPM and associated with session.

Actions

1. The TPM_OIAP command allows the creation of an authorization session handle and the tracking of the handle by the TPM. The TPM generates the handle and nonce.
2. The TPM has an internal limit as to the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. Internally the TPM will do the following:
   a. TPM allocates space to save handle, protocol identification, both nonces and any other information the TPM needs to manage the session.
   b. TPM generates authHandle and nonceEven, returns these to caller
4. On each subsequent use of the OIAP session the TPM MUST generate a new nonceEven value.
5. When TPM_OIAP is wrapped in an encrypted transport session, no input or output parameters are encrypted.

18.1.1 Actions to validate an OIAP session

Start of informative comment:

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OIAP protocol.

Many commands use OIAP authorization. The following description is therefore necessarily abstract.

End of informative comment.

Actions

The TPM MUST perform the following operations:

1. The TPM MUST verify that the authorization session handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TPM_INVALID_AUTHHANDLE
2. The TPM SHALL retrieve the latest version of the caller's nonce (nonceOdd) and continueAuthSession flag from the input parameter list, and store it in internal TPM memory with the authSession 'H'.
3. The TPM SHALL retrieve the latest version of the TPM's nonce stored with the authorization session H (authLastNonceEven) computed during the previously executed command.
4. The TPM MUST retrieve the secret AuthData (SecretE, say) of the target entity. The entity and its secret must have been previously loaded into the TPM.
    a. If the command using the OIAP session requires owner authorization
        i. If TPM_STCLEAR_DATA -> ownerReference is TPM_KH_OWNER, the secret AuthData is TPM_PERMANENT_DATA -> ownerAuth
        ii. If TPM_STCLEAR_DATA -> ownerReference is pointing to a delegate row
            (1) Set R1 a row index to TPM_STCLEAR_DATA -> ownerReference
            (2) Set D1 a TPM_DELEGATE_TABLE_ROW to TPM_PERMANENT_DATA -> delegateTable -> delRow[R1]
            (3) Set the secret AuthData to D1 -> authValue
            (4) Validate the TPM_DELEGATE_PUBLIC D1 -> pub based on the command ordinal
5. The TPM SHALL perform a HMAC calculation using the entity secret data, ordinal, input command parameters and authorization parameters per Part 1 Object-Independent Authorization Protocol.
6. The TPM SHALL compare HM to the AuthData value received in the input parameters. If they are different, the TPM returns the error code TPM_AUTHFAIL if the authorization session is the first session of a command, or TPM_AUTH2FAIL if the authorization session is the second session of a command. Otherwise, the TPM executes the command which (for this example) produces an output that requires authentication.
7. The TPM SHALL generate a nonce (nonceEven).
8. The TPM creates an HMAC digest to authenticate the return code, return values and authorization parameters to the same entity secret per Part 1 Object-Independent Authorization Protocol.
9. The TPM returns the return code, output parameters, authorization parameters and authorization session digest.

10. If the output continueUse flag is FALSE, then the TPM SHALL terminate the session. Future references to H will return an error.

18.2 TPM_OSAP

Start of informative comment:

The TPM_OSAP command creates the authorization session handle, the shared secret and generates nonceEven and nonceEvenOSAP.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM _ORD_OSAP.
4	2			TPM_ENTITY_TYPE	entityType	The type of entity in use
5	4			UINT32	entityValue	The selection value based on entityType, e.g. a keyHandle #
6	20			TPM_NONCE	nonceOddOSAP	The nonce generated by the caller associated with the shared secret.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OSAP.
4	4			TPM_AUTHHANDLE	authHandle	Handle that TPM creates that points to the authorization state.
5	20			TPM_NONCE	nonceEven	Nonce generated by TPM and associated with session.
6	20			TPM_NONCE	nonceEvenOSAP	Nonce generated by TPM and associated with shared secret.

Description

1. The TPM_OSAP command allows the creation of an authorization session handle and the tracking of the handle by the TPM. The TPM generates the handle, nonceEven and nonceEvenOSAP.

2. The TPM has an internal limit on the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.

3. The TPM_OSAP allows the binding of an authorization to a specific entity. This allows the caller to continue to send in AuthData for each command but not have to request the information or cache the actual AuthData.

4. When TPM_OSAP is wrapped in an encrypted transport session, no input or output parameters are encrypted.

5. If the owner pointer is pointing to a delegate row, the TPM internally MUST treat the OSAP session as a DSAP session

6. TPM_ET_SRK or TPM_ET_KEYHANDLE with a value of TPM_KH_SRK MUST specify the SRK.

Actions

1. The TPM creates S1 a storage area that keeps track of the information associated with the authorization.
2. S1 MUST track the following information
    a. Protocol identification (i.e. TPM_PID_OSAP)
    b. nonceEven
        i. Initialized to the next value from the TPM RNG
    c. shared secret
    d. ADIP encryption scheme from TPM_ENTITY_TYPE entityType
    e. Any other internal TPM state the TPM needs to manage the session
3. The TPM MUST create and MAY track the following information
    a. nonceEvenOSAP
        i. Initialized to the next value from the TPM RNG
4. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC calculation is the secret AuthData assigned to the key handle identified by entityValue. The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved in the authorization area associated with authHandle
5. Check if the ADIP encryption scheme specified by entityType is supported, if not return TPM_INAPPROPRIATE_ENC.
6. If entityType = TPM_ET_KEYHANDLE
    a. The entity to authorize is a key held in the TPM. entityValue contains the keyHandle that holds the key.
    b. If entityValue is TPM_KH_OPERATOR return TPM_BAD_HANDLE
7. else if entityType = TPM_ET_OWNER
    a. This value indicates that the entity is the TPM owner. entityValue is ignored
    b. The HMAC key is the secret pointed to by ownerReference (owner secret or delegated secret)
8. else if entityType = TPM_ET_SRK
    a. The entity to authorize is the SRK. entityValue is ignored.
9. else if entityType = TPM_ET_COUNTER
    a. The entity is a monotonic counter, entityValue contains the counter handle
10. else if entityType = TPM_ET_NV
    a. The entity is a NV index, entityValue contains the NV index
11. else return TPM_BAD_PARAMETER

12. On each subsequent use of the OSAP session the TPM MUST generate a new nonce value.

13. The TPM MUST ensure that OSAP shared secret is only available while the OSAP session is valid.

14. The session MUST terminate upon any of the following conditions:

    a. The command that uses the session returns an error

    b. The resource is evicted from the TPM or otherwise invalidated

    c. The session is used in any command for which the shared secret is used to encrypt an input parameter (TPM_ENCAUTH)

    d. The TPM Owner is cleared

    e. TPM_ChangeAuthOwner is executed and this session is attached to the owner authorization

    f. The session explicitly terminated with continueAuth, TPM_Reset or TPM_FlushSpecific

    g. All OSAP sessions MUST be invalidated when any of the following commands execute:

        i. TPM_Delegate_Manage

        ii. TPM_Delegate_CreateOwnerDelegation with Increment==TRUE

        iii. TPM_Delegate_LoadOwnerDelegation

18.2.1 Actions to validate an OSAP session

Start of informative comment:

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OSAP protocol.

Many commands use OSAP authorization. The following description is therefore necessarily abstract.

End of informative comment

Actions

1. On reception of a command with ordinal C1 that uses an authorization session, the TPM SHALL perform the following actions:

2. The TPM MUST have been able to retrieve the shared secret (Shared, say) of the target entity when the authorization session was established with TPM_OSAP. The entity and its secret must have been previously loaded into the TPM.

3. The TPM MUST verify that the authorization session handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TPM_INVALID_AUTHHANDLE.

4. The TPM MUST calculate the HMAC (HM1, say) of the command parameters according to Part 1 Object-Specific Authorization Protocol.

5. The TPM SHALL compare HM1 to the AuthData value received in the command. If they are different, the TPM returns the error code TPM_AUTHFAIL if the authorization session is the first session of a command, or TPM_AUTH2FAIL if the authorization session is the second session of a command., the TPM executes command C1 which produces an output (O, say) that requires authentication and uses a particular return code (RC, say).
6. The TPM SHALL generate the latest version of the even nonce (nonceEven).
7. The TPM MUST calculate the HMAC (HM2) of the return parameters according to section Part 1 Object-Specific Authorization Protocol.
8. The TPM returns HM2 in the parameter list.
9. The TPM SHALL retrieve the continue flag from the received command. If the flag is FALSE, the TPM SHALL terminate the session and destroy the thread associated with handle H.
10. If the shared secret was used to provide confidentiality for data in the received command, the TPM SHALL terminate the session and destroy the thread associated with handle H.
11. Each time that access to an entity (key) is authorized using OSAP, the TPM MUST ensure that the OSAP shared secret is that derived from the entity using TPM_OSAP

18.3 TPM_DSAP

Start of informative comment:

The TPM_DSAP command creates the authorization session handle using a delegated AuthData value passed into the command as an encrypted blob or from the internal delegation table. It can be used to start an authorization session for a user key or the owner.

Identically to TPM_OSAP, it generates a shared secret and generates nonceEven and nonceEvenOSAP.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DSAP.
4	2			TPM_ENTITY_TYPE	entityType	The type of delegation information to use
5	4			TPM_KEY_HANDLE	keyHandle	Key for which delegated authority corresponds, or 0 if delegated owner activity. Only relevant if entityValue equals TPM_DELEGATE_KEY_BLOB
6	20			TPM_NONCE	nonceOddDSAP	The nonce generated by the caller associated with the shared secret.
7	4			UINT32	entityValueSize	The size of entityValue.
8	<>	2S	<>	BYTE [ ]	entityValue	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB or index MUST not be empty If entityType is TPM_ET_DEL_ROW then entityValue is a TPM_DELEGATE_INDEX

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DSAP.
4	4			TPM_AUTHHANDLE	authHandle	Handle that TPM creates that points to the authorization state.
5	20			TPM_NONCE	nonceEven	Nonce generated by TPM and associated with session.
6	20			TPM_NONCE	nonceEvenDSAP	Nonce generated by TPM and associated with shared secret.

Description

1. The TPM_DSAP command allows the creation of an authorization session handle and the tracking of the handle by the TPM. The TPM generates the handle, nonceEven and nonceEvenOSAP.

2. The TPM has an internal limit on the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. The TPM_DSAP allows the binding of a delegated authorization to a specific entity. This allows the caller to continue to send in AuthData for each command but not have to request the information or cache the actual AuthData.
4. Each ordinal that uses the DSAP session MUST validate that TPM_PERMANENT_DATA -> restrictDelegate does not restrict delegation, based on keyHandle -> keyUsage and keyHandle -> keyFlags, return TPM_INVALID_KEYUSAGE on error.
5. On each subsequent use of the DSAP session the TPM MUST generate a new nonce value and check if the ordinal to be executed has delegation to execute. The TPM MUST ensure that the DSAP shared secret is only available while the DSAP session is valid.
6. When TPM_DSAP is wrapped in an encrypted transport session
    a. For input the only parameter encrypted is entityValue
    b. For output no parameters are encrypted
7. The DSAP session MUST terminate under any of the following conditions
    a. The command that uses the session returns an error
    b. If attached to a key, when the key is evicted from the TPM or otherwise invalidated
    c. The session is used in any command for which the shared secret is used to encrypt an input parameter (TPM_ENCAUTH)
    d. The TPM Owner is cleared
    e. TPM_ChangeAuthOwner is executed and this session is attached to the owner authorization
    f. The session explicitly terminated with continueAuth, TPM_Reset or TPM_FlushSpecific
    g. All DSAP sessions MUST be invalidated when any of the following commands execute:
        i. TPM_Delegate_CreateOwnerDelegation
        ii. When Increment is TRUE
        iii. TPM_Delegate_LoadOwnerDelegation
        iv. TPM_Delegate_Manage

entityType = TPM_ET_DEL_OWNER_BLOB

The entityValue parameter contains an owner delegation blob structure.

entityType = TPM_ET_DEL_ROW

The entityValue parameter contains a row number in the nv Delegation table which should be used for the AuthData value.

entityType = TPM_DEL_KEY_BLOB

The entityValue parameter contains a key delegation blob structure.

Actions

```
1. If entityType == TPM_ET_DEL_OWNER_BLOB
    a. Map entityValue to B1 a TPM_DELEGATE_OWNER_BLOB
    b. Validate that B1 is a valid TPM_DELEGATE_OWNER_BLOB, return TPM_WRONG_ENTITYTYPE on error
    c. Locate B1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
    d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
    e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
    f. Verify that B1->verificationCount equals FR -> verificationCount.
    g. Validate the integrity of the blob
        i. Copy B1 -> integrityDigest to H2
        ii. Set B1 -> integrityDigest to NULL
        iii. Create H3 the HMAC of B1 using tpmProof as the secret
        iv. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
    h. Create S1 a TPM_DELEGATE_SENSITIVE by decrypting B1 -> sensitiveArea using TPM_DELEGATE_KEY
    i. Validate S1 values
        i. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE
        ii. Return TPM_BAD_DELEGATE on error
    j. Set A1 to S1 -> authValue
2. Else if entityType == TPM_ET_DEL_ROW
    a. Verify that entityValue points to a valid row in the delegation table.
    b. Set D1 to the delegation information in the row.
    c. Set A1 to D1->authValue.
    d. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that row, return TPM_BADINDEX if not found
    e. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
    f. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
    g. Verify that D1->verificationCount equals FR -> verificationCount.
3. Else if entityType == TPM_ET_DEL_KEY_BLOB
    a. Map entityValue to K1 a TPM_DELEGATE_KEY_BLOB
    b. Validate that K1 is a valid TPM_DELEGATE_KEY_BLOB, return TPM_WRONG_ENTITYTYPE on error
    c. Locate K1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that row, return TPM_BADINDEX if not found
    d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
```

   e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD

   f. Verify that K1 -> pub -> verificationCount equals FR -> verificationCount.

   g. Validate the integrity of the blob

      i. Copy K1 -> integrityDigest to H2

      ii. Set K1 -> integrityDigest to NULL

      iii. Create H3 the HMAC of K1 using tpmProof as the secret

      iv. Compare H2 to H3 return TPM_AUTHFAIL on mismatch

   h. Validate that K1 -> pubKeyDigest identifies keyHandle, return TPM_KEYNOTFOUND on error

   i. Create S1 a TPM_DELEGATE_SENSITIVE by decrypting K1 -> sensitiveArea using TPM_DELEGATE_KEY

   j. Validate S1 values

      i. S1 -> tag is TPM_TAG_DELEGATE_SENSTIVE

      ii. Return TPM_BAD_DELEGATE on error

   k. Set A1 to S1 -> authValue

4. Else return TPM_BAD_PARAMETER

5. Generate a new authorization session handle and reserve space to save protocol identification, shared secret, pcrInfo, both nonces, ADIP encryption scheme, delegated permission bits and any other information the TPM needs to manage the session.

6. Read two new values from the RNG to generate nonceEven and nonceEvenOSAP.

7. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC calculation is A1. The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved in the authorization area associated with authHandle.

18.4 TPM_SetOwnerPointer

Start of informative comment:

This command will set a reference to which secret the TPM will use when executing an owner secret related OIAP or OSAP session.

This command should only be used to provide an owner delegation function for legacy code that does not itself support delegation. Normally, TPM_STCLEAR_DATA->ownerReference points to TPM_KH_OWNER, indicating that OIAP and OSAP sessions should use the owner authorization. This command allows ownerReference to point to an index in the delegation table, indicating that OIAP and OSAP sessions should use the delegation authorization.

In use, a TSS supporting delegation would create and load the owner delegation and set the owner pointer to that delegation. From then on, a legacy TSS application would use its OIAP and OSAP sessions with the delegated owner authorization.

Since this command is not authorized, the ownerReference is open to DoS attacks. Applications can attempt to recover from a failing owner authorization by resetting ownerReference to an appropriate value.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_ COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_SetOwnerPointer
4	2	2S	2	TPM_ENTITY_TYPE	entityType	The type of entity in use
5	4	3S	4	UINT32	entityValue	The selection value based on entityType

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_ COMMAND
2	4			UINT32	paramSize	Total number of output bytes
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_SetOwnerPointer

Actions

1. Map TPM_STCLEAR_DATA to V1
2. If entityType = TPM_ET_DEL_ROW
   a. This value indicates that the entity is a delegate row. entityValue is a delegate index in the delegation table.

b. Validate that entityValue points to a legal row within the delegate table stored within the TPM. If not return TPM_BADINDEX

i. Set D1 to the delegation information in the row.

c. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that row, return TPM_BADINDEX if not found.

d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]

e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD

f. Verify that B1->verificationCount equals FR -> verificationCount.

g. The TPM sets V1-> ownerReference to entityValue

h. Return TPM_SUCCESS

3. else if entityType = TPM_ET_OWNER

a. This value indicates that the entity is the TPM owner. entityValue is ignored.

b. The TPM sets V1-> ownerReference to TPM_KH_OWNER

c. Return TPM_SUCCESS

4. Return TPM_BAD_PARAMETER

19. Delegation Commands

19.1 TPM_Delegate_Manage

Start of informative comment:

TPM_Delegate_Manage is the fundamental process for managing the Family tables, including enabling/disabling Delegation for a selected Family. Normally TPM_Delegate_Manage must be executed at least once (to create Family tables for a particular family) before any other type of Delegation command in that family can succeed.

TPM_Delegate_Manage is authorized by the TPM Owner if an Owner is installed, because changing a table is a privileged Owner operation. If no Owner is installed, TPM_Delegate_Manage requires no privilege to execute. This does not disenfranchise an Owner, since there is no Owner, and simplifies loading of tables during platform manufacture or on first-boot. Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can be locked after loading, to prevent subsequent tampering, and only unlocked by the Owner, his delegate, or the act of removing the Owner (even if there is no Owner).

TPM_Delegate_Manage command is customized by opCode:

(1) TPM_FAMILY_ENABLE enables/disables use of a family and all the rows of the delegate table belonging to that family,

(2) TPM_FAMILY_ADMIN can be used to prevent further management of the Tables until an Owner is installed, or until the Owner is removed from the TPM. (Note that the Physical Presence command TPM_ForceClear always enables further management, even if TPM_ForceClear is used when no Owner is installed.)

(3) TPM_FAMILY_CREATE creates a new family. Sessions are invalidated even in this case because the lastFamilyID could wrap.

(4) TPM_FAMILY_INVALIDATE invalidates an existing family.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_Manage
4	4	2S	4	TPM_FAMILY_ID	familyID	The familyID that is to be managed
5	4	3s	4	TPM_FAMILY_OPERATION	opCode	Operation to be performed by this command.
6	4	4s	4	UINT32	opDataSize	Size in bytes of opData
7	<>	5s	<>	BYTE [ ]	opData	Data necessary to implement opCode
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_Manage
4	4	3S	4	UINT32	retDataSize	Size in bytes of retData
5	<>	4S	<>	BYTE [ ]	retData	Returned data
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	HMAC key: ownerAuth.

Action

1. If opCode != TPM_FAMILY_CREATE
   a. Locate familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
   b. Set FR, a TPM_FAMILY_TABLE_ENTRY, to TPM_FAMILY_TABLE. famTableRow [familyRow]
2. If tag = TPM_TAG_RQU_AUTH1_COMMAND
   a. Validate the command and parameters using ownerAuth, return TPM_AUTHFAIL on error

```
    b. If the authHandle session type is TPM_PID_DSAP
        i. If opCode = TPM_FAMILY_CREATE
            (1) The TPM MUST ignore familyID
        ii. Else
            (1) Verify that the familyID associated with authHandle matches the familyID parameter, return TPM_DELEGATE_FAMILY on error
3. Else
    a. If TPM_PERMANENT_DATA -> ownerAuth is valid, return TPM_AUTHFAIL
    b. If opCode != TPM_FAMILY_CREATE and FR -> flags -> TPM_DELEGATE_ADMIN_LOCK is TRUE, return TPM_DELEGATE_LOCK
    c. Validate max NV writes without an owner
        i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite
        ii. Increment NV1 by 1
        iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
        iv. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1
4. The TPM invalidates sessions
    a. MUST invalidate all DSAP sessions
    b. MUST invalidate all OSAP sessions associated with the delegation table
    c. MUST set TPM_STCLEAR_DATA -> ownerReference to TPM_KH_OWNER
    d. MAY invalidate any other session
5. If opCode == TPM_FAMILY_CREATE
    a. Validate that sufficient space exists within the TPM to store an additional family and map F2 to the newly allocated space.
    b. Validate that opData is a TPM_FAMILY_LABEL
        i. If opDataSize != sizeof(TPM_FAMILY_LABEL) return TPM_BAD_PARAM_SIZE
    c. Map F2 to a TPM_FAMILY_TABLE_ENTRY
        i. Set F2 -> tag to TPM_TAG_FAMILY_TABLE_ENTRY
        ii. Set F2 -> familyLabel to opData
    d. Increment TPM_PERMANENT_DATA -> lastFamilyID by 1
    e. Set F2 -> familyID = TPM_PERMANENT_DATA -> lastFamilyID
    f. Set F2 -> verificationCount = 1
    g. Set F2 -> flags -> TPM_FAMFLAG_ENABLED to FALSE
    h. Set F2 -> flags -> TPM_DELEGATE_ADMIN_LOCK to FALSE
    i. Set retDataSize = 4
    j. Set retData = F2 -> familyID
    k. Return TPM_SUCCESS
6. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE
7. If opCode == TPM_FAMILY_ADMIN
    a. Validate that opDataSize == 1, and that opData is a Boolean value.
    b. Set (FR -> flags -> TPM_DELEGATE_ADMIN_LOCK) = opData
    c. Set retDataSize = 0
    d. Return TPM_SUCCESS
8. else If opCode == TPM_FAMILY_ENABLE
    a. Validate that opDataSize == 1, and that opData is a Boolean value.
    b. Set FR -> flags-> TPM_FAMFLAG_ENABLED = opData
    c. Set retDataSize = 0
    d. Return TPM_SUCCESS
9. else If opCode == TPM_FAMILY_INVALIDATE
    a. Invalidate all data associated with familyRow
        i. All data is all information pointed to by FR
        ii. return TPM_SELFTEST_FAILED on failure
    b. Set retDataSize = 0
    c. Return TPM_SUCCESS
10. Else return TPM_BAD_PARAMETER
```

19.2 TPM_Delegate_CreateKeyDelegation

Start of informative comment:

This command delegates privilege to use a key by creating a blob that can be used by TPM_DSAP.

There is no check for appropriateness of the key's key usage against the key permission settings. If the key usage is incorrect, this command succeeds, but the delegated command will fail.

These blobs CANNOT be used as input data for TPM_LoadOwnerDelegation because the internal TPM delegate table can store owner delegations only.

(TPM_Delegate_CreateOwnerDelegation must be used to delegate Owner privilege.)

End of informative comment

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_CreateKeyDelegation.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key.
5	<>	2S	<>	TPM_DELEGATE_PUBLIC	publicInfo	The public information necessary to fill in the blob
6	20	3S	20	TPM_ENCAUTH	delAuth	The encrypted new AuthData for the blob. The encryption key is the shared secret from the OSAP protocol.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Ignored
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the use of keyHandle. HMAC key: key.usageAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Delegate_CreateKeyDelegation
4	4	3S	4	UINT32	blobSize	The length of the returned blob
5	<>	4S	<>	TPM_DELEGATE_KEY_BLOB	blob	The partially encrypted delegation information.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag. Fixed value of FALSE
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

Description

1. The use restrictions that may be present on the key pointed to by keyHandle are not enforced for this command. Stated another way, TPM_CreateKeyDelegation is not a use of the key.

Action

1. Verify AuthData for the command and parameters using privAuth
2. Locate publicInfo -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
3. If the key authentication is in fact a delegation, then the TPM SHALL validate the command and parameters using Delegation authorisation, then
    a. Validate that authHandle -> familyID equals publicInfo -> familyID return TPM_DELEGATE_FAMILY on error
    b. If TPM_FAMILY_TABLE.famTableRow[authHandle -> familyID] -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return error TPM_DISABLED_CMD.
    c. Verify that the delegation bits in publicInfo do not grant more permissions then currently delegated. Otherwise return error TPM_AUTHFAIL
4. Check that publicInfo -> delegateType is TPM_DEL_KEY_BITS
5. Verify that authHandle indicates an OSAP or DSAP session return TPM_INVALID_AUTHHANDLE on error
6. If authHandle indicates XOR encryption for the AuthData secrets
    a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
    b. Create a1 by XOR X1 and delAuth
7. Else

   a. Create a1 by decrypting delAuth using the algorithm indicated in the OSAP session

   b. Key is from authHandle -> sharedSecret

   c. IV is SHA-1 of (authLastNonceEven || nonceOdd)

8. Create h1 the SHA-1 of TPM_STORE_PUBKEY structure of the key pointed to by keyHandle

9. Create M1 a TPM_DELEGATE_SENSITIVE structure

   a. Set M1 -> tag to TPM_TAG_DELEGATE_SENSITIVE

   b. Set M1 -> authValue to a1

   c. The TPM MAY add additional information of a sensitive nature relative to the delegation

10. Create M2 the encryption of M1 using TPM_DELEGATE_KEY

11. Create P1 a TPM_DELEGATE_KEY_BLOB

   a. Set P1 -> tag to TPM_TAG_DELG_KEY_BLOB

   b. Set P1 -> pubKeyDigest to H1

   c. Set P1 -> pub to PublicInfo

   d. Set P1 -> pub -> verificationCount to familyRow -> verificationCount

   e. Set P1 -> integrityDigest to NULL

   f. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The information MAY include symmetric IV, symmetric mode of encryption and other data that allows the TPM to process the blob in the future.

   g. Set P1 -> sensitiveSize to the size of M2

   h. Set P1 -> sensitiveArea to M2

12. Calculate H2 the HMAC of P1 using tpmProof as the secret

13. Set P1 -> integrityDigest to H2

14. Ignore continueAuthSession on input set continueAuthSession to FALSE on output

15. Return P1 as blob

19.3 TPM_Delegate_CreateOwnerDelegation

Start of informative comment:

TPM_Delegate_CreateOwnerDelegation delegates the Owner's privilege to use a set of command ordinals, by creating a blob. Such blobs can be used as input data for TPM_DSAP or TPM_Delegate_LoadOwnerDelegation.

TPM_Delegate_CreateOwnerDelegation includes the ability to void all existing delegations (by incrementing the verification count) before creating the new delegation. This ensures that the new delegation will be the only delegation that can operate at Owner privilege in this family. This new delegation could be used to enable a security monitor (a local separate entity, or remote separate entity, or local host entity) to reinitialize a family and perhaps perform external verification of delegation settings. Normally the ordinals for a delegated security monitor would include TPM_Delegate_CreateOwnerDelegation (this command) in order to permit the monitor to create further delegations, and TPM_Delegate_UpdateVerification to reactivate some previously voided delegations.

If the verification count is incremented and the new delegation does not delegate any privileges (to any ordinals) at all, or uses an authorisation value that is then discarded, this family's delegations are all void and delegation must be managed using actual Owner authorisation.

(TPM_Delegate_CreateKeyDelegation must be used to delegate privilege to use a key.)

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_Delegate_CreateOwnerDelegation.
4	1	2S	1	BOOL	increment	Flag dictates whether verificationCount will be incremented
5	<>	3S	<>	TPM_DELEGATE_PUBLIC	publicInfo	The public parameters for the blob
6	20	4S	20	TPM_ENCAUTH	delAuth	The encrypted new AuthData for the blob. The encryption key is the shared secret from the OSAP protocol.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Ignored
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_Delegate_CreateOwnerDelegation
4	4	3S	4	UINT32	blobSize	The length of the returned blob
5	<>	4S	<>	TPM_DELEGATE_OWNER _BLOB	blob	The partially encrypted delegation information.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag. Fixed value of FALSE
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth

Action

1. The TPM SHALL authenticate the command using TPM Owner authentication. Return TPM_AUTHFAIL on failure.
2. Locate publicInfo -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate the row return TPM_BADINDEX if not found
    a. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
3. If the TPM Owner authentication is in fact a delegation
    a. Validate that authHandle -> familyID equals publicInfo -> familyID return TPM_DELEGATE_FAMILY on error
    b. If FR -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return error TPM_DISABLED_CMD.
    c. Verify that the delegation bits in publicInfo do not grant more permissions then currently delegated. Otherwise, return error TPM_AUTHFAIL.
4. Check that publicInfo -> delegateType is TPM_DEL_OWNER_BITS
5. Verify that authHandle indicates an OSAP or DSAP session return TPM_INVALID_AUTHHANDLE on error
6. If increment == TRUE
    a. Increment FR -> verificationCount
    b. Set TPM_STCLEAR_DATA-> ownerReference to TPM_KH_OWNER
    c. The TPM invalidates sessions
        i. MUST invalidate all DSAP sessions
        ii. MUST invalidate all OSAP sessions associated with the delegation table

        iii. MAY invalidate any other session

7. If authHandle indicates XOR encryption for the AuthData secrets
    a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
    b. Create a1 by XOR X1 and delAuth
8. Else
    a. Create a1 by decrypting delAuth using the algorithm indicated in the OSAP session
    b. Key is from authHandle -> sharedSecret
    c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
9. Create M1 a TPM_DELEGATE_SENSITIVE structure
    a. Set M1 -> tag to TPM_TAG_DELEGATE_SENSITIVE
    b. Set M1 -> authValue to a1
    c. Set other M1 fields as determined by the TPM vendor
10. Create M2 the encryption of M1 using TPM_DELEGATE_KEY
11. Create B1 a TPM_DELEGATE_OWNER_BLOB
    a. Set B1 -> tag to TPM_TAG_DELG_OWNER_BLOB
    b. Set B1 -> pub to publicInfo
    c. Set B1 -> sensitiveSize to the size of M2
    d. Set B1 -> sensitiveArea to M2
    e. Set B1 -> integrityDigest to NULL
    f. Set B1 -> pub -> verificationCount to FR -> verificationCount
12. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The information MAY include symmetric IV, symmetric mode of encryption and other data that allows the TPM to process the blob in the future.
13. Create H1 the HMAC of B1 using tpmProof as the secret
14. Set B1 -> integrityDigest to H1
15. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
16. Return B1 as blob

19.4 TPM_Delegate_LoadOwnerDelegation

Start of informative comment:

This command loads a delegate table row blob into a non-volatile delegate table row. TPM_Delegate_LoadOwnerDelegation can be used during manufacturing or on first boot (when no Owner is installed), or after an Owner is installed. If an Owner is installed, TPM_Delegate_LoadOwnerDelegation requires Owner authorisation, and sensitive information must be encrypted.

Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can be locked after loading using TPM_Delegate_Manage, to prevent subsequent tampering.

A management system outside the TPM is expected to manage the delegate table rows stored on the TPM, and can overwrite any previously stored data.

This command cannot be used to load key delegation blobs into the TPM

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_Delegate_LoadOwnerDelegation
4	4	3S	4	TPM_DELEGATE_INDEX	index	The index of the delegate row to be written
5	4	4S	4	UINT32	blobSize	The size of the delegate blob
6	<>	5S	<>	TPM_DELEGATE_OWNER_BLOB	blob	Delegation information, including encrypted portions as appropriate
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_Delegate_LoadOwnerDelegation
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: ownerAuth.

Actions

1. Map blob to D1 a TPM_DELEGATE_OWNER_BLOB.
   a. Validate that D1 -> tag == TPM_TAG_DELEGATE_OWNER_BLOB
2. Locate D1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
3. Set FR to TPM_FAMILY_TABLE -> famTableRow[familyRow]
4. If TPM Owner is installed
   a. Validate the command and parameters using TPM Owner authentication, return TPM_AUTHFAIL on error
   b. If the authHandle session type is TPM_PID_DSAP, verify that D1 -> pub -> familyID matches authHandle -> familyID, on error return TPM_DELEGATE_FAMILY
5. Else
   a. If FR -> flags -> TPM_DELEGATE_ADMIN_LOCK is TRUE return TPM_DELEGATE_LOCK
   b. Validate max NV writes without an owner
      i. Set NV1 to PD -> noOwnerNVWrite
      ii. Increment NV1 by 1
      iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
      iv. Set PD -> noOwnerNVWrite to NV1
6. If FR -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
7. If TPM Owner is installed, validate the integrity of the blob
   a. Copy D1 -> integrityDigest to H2
   b. Set D1 -> integrityDigest to NULL
   c. Create H3 the HMAC of D1 using tpmProof as the secret

    d. Compare H2 to H3, return TPM_AUTHFAIL on mismatch

8. If TPM Owner is installed, create S1 a TPM_DELEGATE_SENSITIVE area by decrypting D1 -> sensitiveArea using TPM_DELEGATE_KEY. Otherwise set S1 = D1 -> sensitiveArea

9. Validate S1

    a. Validate that S1 -> tag == TPM_TAG_DELEGATE_SENSITIVE, return TPM_INVALID_STRUCTURE on error

10. Validate that index is a valid value for delegateTable, return TPM_BADINDEX on error

11. The TPM invalidates sessions

    a. MUST invalidate all DSAP sessions

    b. MUST invalidate all OSAP sessions associated with the delegation table

    c. MAY invalidate any other session

12. Copy data to the delegate table row

    a. Copy the TPM_DELEGATE_PUBLIC from D1 -> pub to TPM_DELEGATE_TABLE -> delRow[index] -> pub.

    b. Copy the TPM_SECRET from S1 -> authValue to TPM_DELEGATE_TABLE -> delRow[index] -> authValue.

    c. Set TPM_STCLEAR_DATA-> ownerReference to TPM_KH_OWNER

    d. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE

13. Return TPM_SUCCESS

19.5 TPM_Delegate_ReadTable

Start of informative comment:

This command reads from the TPM the public contents of the family and delegate tables that are stored on the TPM. Such data is required during external verification of tables.

There are no restrictions on the execution of this command; anyone can read this information regardless of the state of the PCRs, regardless of whether they know any specific AuthData value and regardless of whether or not the enable and admin bits are set one way or the other.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Delegate_ReadTable

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Delegate_ReadTable
4	4	3S	4	UINT32	familyTableSize	Size in bytes of familyTable
5	<>	4S	<>	BYTE [ ]	familyTable	Array of TPM_FAMILY_TABLE_ENTRY elements
6	4	5S	4	UINT32	delegateTableSize	Size in bytes of delegateTable
7	<>	6S	<>	BYTE[]	delegateTable	Array of TPM_DELEGATE_INDEX and TPM_DELEGATE_PUBLIC elements

Actions

1. Set familyTableSize to the number of valid families on the TPM times sizeof(TPM_FAMILY_TABLE_ELEMENT).
2. Copy the valid entries in the internal family table to the output array familyTable
3. Set delegateTableSize to the number of valid delegate table entries on the TPM times (sizeof(TPM_DELEGATE_PUBLIC) + 4).
4. For each valid entry
   a. Write the TPM_DELEGATE_INDEX to delegateTable
   b. Copy the TPM_DELEGATE_PUBLIC to delegateTable

5. Return TPM_SUCCESS

19.6 TPM_Delegate_UpdateVerification

Start of informative comment:

TPM_UpdateVerification sets the verificationCount in an entity (a blob or a delegation row) to the current family value, in order that the delegations represented by that entity will continue to be accepted by the TPM.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_UpdateVerification
4	4	2S	4	UINT32	inputSize	The size of inputData
5	<>	3S	<>	BYTE	inputData	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_INDEX
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEv en	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	ownerAuth	Authorization HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_UpdateVerification
4	4 .	3S	4	UINT32	outputSize	The size of the output
5	<>	4S	<>	BYTE	outputData	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Actions

1. Verify the TPM Owner, directly or indirectly through delegation, authorizes the command and parameters, on error return TPM_AUTHFAIL
2. Determine the type of inputData (TPM_DELEGATE_TABLE_ROW or TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB) and map D1 to that structure
    a. Mapping to TPM_DELEGATE_TABLE_ROW requires taking inputData as a tableIndex and locating the appropriate row in the table
3. If D1 is a TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB, validate the integrity of D1
    a. Copy D1 -> integrityDigest to H2
    b. Set D1 -> integrityDigest to NULL
    c. Create H3 the HMAC of D1 using tpmProof as the secret
    d. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
4. Locate (D1 -> pub -> familyID) in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
5. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
6. If delegated, verify that family of the delegated Owner-auth is the same as D1: (authHandle -> familyID) == (D1 -> pub -> familyID); otherwise return error TPM_DELEGATE_FAMILY
7. If delegated, verify that the family of the delegated Owner-auth is enabled: if (authHandle -> familyID -> flags TPM_FAMFLAG_ENABLED) is FALSE, return TPM_DISABLED_CMD
8. Set D1 -> verificationCount to FR -> verificationCount

9. If D1 is a TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB set the integrity of D1
    a. Set D1 -> integrityDigest to NULL
    b. Create H1 the HMAC of D1 using tpmProof as the secret
    c. Set D1 -> integrityDigest to H1
10. If D1 is a blob recreate the blob and return it

19.7 TPM_Delegate_VerifyDelegation

Start of informative comment:

TPM_VerifyDelegation interprets a delegate blob and returns success or failure, depending on whether the blob is currently valid. The delegate blob is NOT loaded into the TPM.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal, TPM_Delegate_VerifyDelegation
4	4	2S	4	UINT32	delegationSize	The length of the delegated information blob
5	<>	3S	<>	BYTE[ ]	delegation	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal, TPM_Delegate_VerifyDelegation

Actions

1. Determine the type of blob, If delegation -> tag is equal to TPM_TAG_DELGATE_OWNER_BLOB then
   a. Map D1 a TPM_DELEGATE_OWNER_BLOB to delegation
2. Else if delegation -> tag = TPM_TAG_DELG_KEY_BLOB
   a. Map D1 a TPM_DELEGATE_KEY_BLOB to delegation
3. Else return TPM_BAD_PARAMETER
4. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
5. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
6. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
7. Validate that D1 -> pub -> verificationCount matches FR -> verificationCount, on mismatch return TPM_FAMILYCOUNT
8. Validate the integrity of D1
   a. Copy D1 -> integrityDigest to H2

   b. Set D1 -> integrityDigest to NULL
   c. Create H3 the HMAC of D1 using tpmProof as the secret
   d. Compare H2 to H3 return TPM_AUTHFAIL on mismatch

9. Create S1 a TPM_DELEGATE_SENSITIVE area by decrypting D1 -> sensitiveArea using TPM_DELEGATE_KEY

10. Validate S1 values

   a. S1 -> tag is TPM_TAG_DELEGATE_SENSTIVE
   b. Return TPM_BAD_PARAMETER on error

11. Return TPM_SUCCESS

20. Non-volatile Storage

Start of informative comment:

This section handles the allocation and use of the TPM non-volatile storage.

End of informative comment.

If nvIndex refers to the DIR, the TPM ignores actions containing access control checks that have no meaning for the DIR. The TPM only checks the owner authorization.

20.1 TPM_NV_DefineSpace

Start of informative comment:

This establishes the space necessary for the indicated index. The definition will include the access requirements for writing and reading the area.

The space definition size does not include the area needed to manage the space.

Setting TPM_PERMANENT_FLAGS -> nvLocked TRUE when it is already TRUE is not an error.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_DefineSpace
4	<>	2S	<>	TPM_NV_DATA_PUBLIC	pubInfo	The public parameters of the NV area
5	20	3S	20	TPM_ENCAUTH	encAuth	The encrypted AuthData, only valid if the attributes require subsequent authorization
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for ownerAuth
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal, TPM_ORD_NV_DefineSpace
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, fixed to FALSE
6	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

Actions

1. If pubInfo -> nvIndex == TPM_NV_INDEX_LOCK and tag = TPM_TAG_RQU_COMMAND

a. If pubInfo -> dataSize is not 0, the command MAY return TPM_BADINDEX.

b. Set TPM_PERMANENT_FLAGS -> nvLocked to TRUE

c. Return TPM_SUCCESS

2. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks except for the Max NV writes are ignored

a. Ignored checks include physical presence, authorization, 'D' bit check, index 0, bGlobalLock, no authorization with a TPM owner present, and bWriteSTClear

3. If pubInfo -> nvIndex has the D bit (bit 28) set to a 1 or pubInfo -> nvIndex == 0 then

a. Return TPM_BADINDEX

b. The D bit specifies an index value that is set in manufacturing and can never be deleted or added to the TPM

c. Index value of 0 is reserved and cannot be defined

4. If tag = TPM_TAG_RQU_AUTH1_COMMAND then

a. The TPM MUST validate the command and parameters using the TPM Owner authentication and ownerAuth, on error return TPM_AUTHFAIL

b. authHandle session type MUST be OSAP

c. If authHandle indicates XOR encryption for the AuthData secrets

i. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)

ii. Create a1 by XOR X1 and encAuth

d. Else

i. Create a1 by decrypting encAuth using the algorithm indicated in the OSAP session

ii. Key is from authHandle -> sharedSecret

iii. IV is SHA-1 of (authLastNonceEven || nonceOdd)

5. else

a. Validate the assertion of physical presence. Return TPM_BAD_PRESENCE on error.

b. If TPM Owner is present then return TPM_OWNER_SET.

c. If pubInfo -> dataSize is 0 then return TPM_BAD_DATASIZE. Setting the size to 0 represents an attempt to delete the value without TPM Owner authentication.

d. Validate max NV writes without an owner

i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite

ii. Increment NV1 by 1

iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES

iv. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1

e. Set A1 to encAuth. There is no nonce or authorization to create the encryption string, hence the AuthData value is passed in the clear

6. If pubInfo -> nvIndex points to a valid previously defined storage area then

a. Map D1 a TPM_NV_DATA_SENSITIVE to the storage area

b. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK then

i. If TPM_STCLEAR_FLAGS -> bGlobalLock is TRUE then return TPM_AREA_LOCKED

c. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR

i. If D1 -> pubInfo -> bWriteSTClear is TRUE then return TPM_AREA_LOCKED

d. Invalidate the data area currently pointed to by D1 and ensure that if the area is reallocated no residual information is left

e. The TPM invalidates authorization sessions

i. MUST invalidate all authorization sessions associated with D1

ii. MAY invalidate any other authorization session

f. If pubInfo -> dataSize is 0 then return TPM_SUCCESS

7. Parse pubInfo -> pcrInfoRead

a. Validate pcrInfoRead structure on error return TPM_INVALID_STRUCTURE

i. Validation includes proper PCR selections and locality selections

8. Parse pubInfo -> pcrInfoWrite

a. Validate pcrInfoWrite structure on error return TPM_INVALID_STRUCTURE

i. Validation includes proper PCR selections and locality selections

b. If pcrInfoWrite -> localityAtRelease disallows some localities

i. Set writeLocalities to TRUE

c. Else

i. Set writeLocalities to FALSE

9. Validate that the attributes are consistent

a. The TPM SHALL ignore the bReadSTClear, bWriteSTClear and bWriteDefine attributes during the execution of this command

b. If TPM_NV_PER_OWNERWRITE is TRUE and TPM_NV_PER_AUTHWRITE is TRUE return TPM_AUTH_CONFLICT

c. If TPM_NV_PER_OWNERREAD is TRUE and TPM_NV_PER_AUTHREAD is TRUE return TPM_AUTH_CONFLICT

d. If TPM_NV_PER_OWNERWRITE and TPM_NV_PER_AUTHWRITE and TPM_NV_PER_WRITEDEFINE and TPM_NV_PER_PPWRITE and writeLocalities are all FALSE

i. Return TPM_PER_NOWRITE

e. Validate nvIndex

i. Make sure that the index is applicable for this TPM return TPM_BADINDEX on error. A valid index is platform and context sensitive. That is attempting to validate an index may be successful in one configuration and invalid in another configuration. The individual index values MUST indicate if there are any restrictions on the use of the index.

f. If dataSize is 0 return TPM_BAD_PARAM_SIZE

10. Create D1 a TPM_NV_DATA_SENSITIVE structure

11. Validate that sufficient NV is available to store the data

a. return TPM_NOSPACE if pubInfo -> dataSize is not available in the TPM

12. Ensure that the TPM reserves the space for dataSize

a. Set all bytes in the newly defined area to 0xFF

13. Set D1 -> pubInfo to pubInfo

14. Set D1 -> authValue to A1

15. Set D1 -> pubInfo -> bReadSTClear = FALSE;

16. Set D1 -> pubInfo -> bWriteSTClear = FALSE;

17. Set D1 -> pubInfo -> bWriteDefine = FALSE;

18. Ignore continueAuthSession on input and set to FALSE on output

19. Return TPM_SUCCESS

20.2 TPM_NV_WriteValue

Start of informative comment:

This command writes the value to a defined area. The write can be TPM Owner authorized or unauthorized and protected by other attributes and will work when no TPM Owner is present.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_WriteValue
4	4	2S	4	TPM_NV_INDEX	nvIndex	The index of the area to set
5	4	3S	4	UINT32	offset	The offset into the NV Area
6	4	4S	4	UINT32	dataSize	The size of the data parameter
7	<>	5S	<>	BYTE	data	The data to set the area to
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for TPM Owner
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal, TPM_ORD_NV_WriteValue
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

Actions

1. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks except for the max NV writes are ignored
    a. Ignored checks include physical presence, authorization, TPM_NV_PER_OWNERWRITE, and PCR

2. If nvIndex = 0 then
    - a. If dataSize is not 0, return TPM_BADINDEX.
    - b. Set TPM_STCLEAR_FLAGS -> bGlobalLock to TRUE
    - c. Return TPM_SUCCESS
3. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, return TPM_BADINDEX on error
    - a. If nvIndex= TPM_NV_INDEX_DIR, set D1 to TPM_PERMANENT_DATA -> authDir[0]
4. If D1 -> permission -> TPM_NV_PER_AUTHWRITE is TRUE return TPM_AUTH_CONFLICT
5. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
    - a. If D1 -> permission -> TPM_NV_PER_OWNERWRITE is FALSE return TPM_AUTH_CONFLICT
    - b. Validate command and parameters using ownerAuth HMAC with TPM Owner authentication as the secret, return TPM_AUTHFAIL on error
6. Else
    - a. If D1 -> permission -> TPM_NV_PER_OWNERWRITE is TRUE return TPM_AUTH_CONFLICT
    - b. If no TPM Owner validate max NV writes without an owner
        - i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite
        - ii. Increment NV1 by 1
        - iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
        - iv. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1
7. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
    - a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
    - b. On error return TPM_BAD_LOCALITY
8. If D1 -> attributes specifies TPM_NV_PER_PPWRITE then validate physical presence is asserted if not return TPM_BAD_PRESENCE
9. If D1 -> attributes specifies TPM_NV_PER_WRITEDEFINE
    - a. If D1 -> bWriteDefine is TRUE return TPM_AREA_LOCKED
10. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK
    - a. If TPM_STCLEAR_DATA -> bGlobalLock is TRUE return TPM_AREA_LOCKED
11. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
    - a. If D1 ->bWriteSTClear is TRUE return TPM_AREA_LOCKED
12. If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of TPM_STCLEAR_DATA -> PCR[]

a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 -> pcrInfoWrite

b. Compare P1 to D1 -> pcrInfoWrite -> digestAtRelease return TPM_WRONGPCRVAL on mismatch

13. If dataSize = 0 then

    a. Set D1 -> bWriteSTClear to TRUE

    b. Set D1 -> bWriteDefine to TRUE

14. Else

    a. Set S1 to offset + dataSize

    b. If S1 > D1 -> dataSize return TPM_NOSPACE

    c. If D1 -> attributes specifies TPM_NV_PER_WRITEALL

        i. If dataSize != D1 -> dataSize return TPM_NOT_FULLWRITE

    d. Write the new value into the NV storage area

15. Set D1 -> bReadSTClear to FALSE

16. Return TPM_SUCCESS

20.3 TPM_NV_WriteValueAuth

Start of informative comment:

This command writes to a previously defined area. The area must require authorization to write. Use this command when authorization other than the owner authorization is to be used. Otherwise, use TPM_NV_WriteValue.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_WriteValueAuth
4	4	2S	4	TPM_NV_INDEX	nvIndex	The index of the area to set
5	4	3S	4	UINT32	offset	The offset into the chunk
6	4	4S	4	UINT32	dataSize	The size of the data area
7	<>	5S	<>	BYTE	data	The data to set the area to
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for NV element authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	authValue	HMAC key: NV element auth value

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal, TPM_ORD_N V_WriteValueAuth
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	NonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	authValue	HMAC key: NV element auth value

Actions

1. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, return TPM_BADINDEX on error
2. If D1 -> attributes does not specify TPM_NV_PER_AUTHWRITE then return TPM_AUTH_CONFLICT

```
3. Validate authValue using D1 -> authValue, return TPM_AUTHFAIL on error
4. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
   a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
   b. On error return TPM_BAD_LOCALITY
5. If D1 -> attributes specifies TPM_NV_PER_PPWRITE then validate physical presence is asserted if not return TPM_BAD_PRESENCE
6. If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of PCR
   a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 -> pcrInfoWrite
   b. Compare P1 to digestAtRelease return TPM_WRONGPCRVAL on mismatch
7. If D1 -> attributes specifies TPM_NV_PER_WRITEDEFINE
   a. If D1 -> bWriteDefine is TRUE return TPM_AREA_LOCKED
8. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK
   a. If TPM_STCLEAR_FLAGS -> bGlobalLock is TRUE return TPM_AREA_LOCKED
9. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
   a. If D1 -> bWriteSTClear is TRUE return TPM_AREA_LOCKED
10. If dataSize = 0 then
   a. Set D1 -> bWriteSTClear to TRUE
   b. Set D1 -> bWriteDefine to TRUE
11. Else
   a. Set S1 to offset + dataSize
   b. If S1 > D1 -> dataSize return TPM_NOSPACE
   c. If D1 -> attributes specifies TPM_NV_PER_WRITEALL
      i. If dataSize != D1 -> dataSize return TPM_NOT_FULLWRITE
   d. Write the new value into the NV storage area
12. Set D1 -> bReadSTClear to FALSE
13. Return TPM_SUCCESS
```

20.4 TPM_NV_ReadValue

Start of informative comment:

Read a value from the NV store. This command uses optional owner authentication.

Action 1 indicates that if the NV are is not locked then reading of the NV area continues without ANY authorization. This is intentional and allows a platform manufacturer to set the NV areas, read them back, and then lock them all without having to install a TPM owner.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_ReadValue
4	4	2S	4	TPM_NV_INDEX	nvIndex	The index of the area to set
5	4	3S	4	UINT32	offset	The offset into the area
6	4	4S	4	UINT32	dataSize	The size of the data area
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for TPM Owner authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S		TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_NV_ReadValue
4	4	3S	4	UINT32	dataSize	The size of the data area
5	<>	4S	<>	BYTE	data	The data to set the area to
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth

Actions

1. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks are ignored
2. Set D1 a TPM_NV_DATA_AREA structure to the area pointed to by nvIndex, if not found return TPM_BADINDEX
    a. If nvIndex = TPM_NV_INDEX_DIR, set D1 to TPM_PERMANENT_DATA -> authDir[0]
3. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
    a. If D1 -> TPM_NV_PER_OWNERREAD is FALSE return TPM_AUTH_CONFLICT
    b. Validate command and parameters using TPM Owners authentication on error return TPM_AUTHFAIL
4. Else
    a. If D1 -> TPM_NV_PER_AUTHREAD is TRUE return TPM_AUTH_CONFLICT
    b. If D1 -> TPM_NV_PER_OWNERREAD is TRUE return TPM_AUTH_CONFLICT
5. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
    a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
    b. On error return TPM_BAD_LOCALITY
6. If D1 -> attributes specifies TPM_NV_PER_PPREAD then validate physical presence is asserted if not return TPM_BAD_PRESENCE
7. If D1 -> TPM_NV_PER_READ_STCLEAR then
    a. If D1 -> bReadSTClear is TRUE return TPM_DISABLED_CMD
8. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR
    a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 -> pcrInfoRead
    b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM_WRONGPCRVAL on mismatch
9. If dataSize is 0 then
    a. Set D1 -> bReadSTClear to TRUE
    b. Set data to NULL
10. Else
    a. Set S1 to offset + dataSize
    b. If S1 > D1 -> dataSize return TPM_NOSPACE
    c. Set data to area pointed to by offset
11. Return TPM_SUCCESS

20.5 TPM_NV_ReadValueAuth

Start of informative comment:

This command requires that the read be authorized by a value set with the blob.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, TPM_ORD_NV_ReadValueAuth
4	4	2S	4	TPM_NV_INDEX	nvIndex	The index of the area to set
5	4	3S	4	UNIT32	offset	The offset from the data area
6	4	5S	4	UINT32	dataSize	The size of the data area
7	4			TPM_AUTHHANDLE	authHandle	authThe auth handle for the NV element authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	authContinueSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	authHmac	HMAC key: nv element authorization

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal, TPM_ORD_NV_ReadValueAuth
4	4	3S	4	UINT32	dataSize	The size of the data area
5	<>	4S	<>	BYTE	data	The data
6	20	2H1	20	TPM_NONCE	authNonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	authLastNonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	authContinueSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	authHmacOut	HMAC key: nv element authorization

Actions

1. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, on error return TPM_BADINDEX
2. If D1 -> TPM_NV_PER_AUTHREAD is FALSE return TPM_AUTH_CONFLICT
3. Validate authHmac using D1 -> authValue on error return TPM_AUTHFAIL

4. If D1 -> attributes specifies TPM_NV_PER_PPREAD then validate physical presence is asserted if not return TPM_BAD_PRESENCE
5. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
    a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
    b. On error return TPM_BAD_LOCALITY
6. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR
    a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 -> pcrInfoRead
    b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM_WRONGPCRVAL on mismatch
7. If D1 specifies TPM_NV_PER_READ_STCLEAR then
    a. If D1 -> bReadSTClear is TRUE return TPM_DISABLED_CMD
8. If dataSize is 0 then
    a. Set D1 -> bReadSTClear to TRUE
    b. Set data to NULL
9. Else
    a. Set S1 to offset + dataSize
    b. If S1 > D1 -> dataSize return TPM_NOSPACE
    c. Set data to area pointed to by offset
10. Return TPM_SUCCESS

21. Session Management

Start of informative comment:

Three TPM_RT_CONTEXT session resources located in TPM_STANY_DATA work together to control session save and load: contextNonceSession, contextCount, and contextList[].

All three MUST initialized at TPM_Startup(ST_ANY), which invalidates all saved sessions. They MAY be restored by TPM_Startup(ST_STATE), which would allow saved sessions to be loaded. The operation is reported as the TPM_RT_CONTEXT startup effect.

TPM_SaveContext creates a contextBlob containing an encrypted contextNonceSession. The nonce is checked by TPM_LoadContext. So initializing contextNonceSession invalidates all saved contexts. The nonce is large and protected, making a replay infeasible.

The contextBlob also contains a public but protected contextCount. The count increments for each saved contextBlob. The TPM also saves contextCount in contextList[]. The TPM validates contextBlob against the contextList[] during TPM_LoadContext. Since the contextList[] is finite, it limits the number of valid saved sessions. Since the contextCount cannot be allowed to wrap, it limits the total number of saved sessions.

After a contextBlob is loaded, its contextCount entry is removed from contextList[]. This releases space in the context list for future entries. It also invalidates the contextBlob. So a saved contextBlob can be loaded only once.

TPM_FlushSpecific can also specify a contextCount to be removed from the contextList[], allowing invalidation of an individual contextBlob. This is different from TPM_FlushSpecific specifying a session handle, which invalidates a loaded session, not a saved contextBlob.

End of informative comment.

21.1 TPM_KeyControlOwner

Start of informative comment:

This command controls some attributes of keys that are stored within the TPM key cache.

OwnerEvict: If this bit is set to true, this key remains in the TPM through all TPM_Startup events. The only way to evict this key is for the TPM Owner to execute this command again, setting the owner control bit to false and then executing TPM_FlushSpecific.

The key handle does not reference an authorize d entity and is not validated.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_KeyControlOwner
4	4			TPM_KEY_HANDLE	keyHandle	The handle of a loaded key.

5	<>	2S	<>	TPM_PUBKEY	pubKey	The public key associated with the loaded key
6	4	3S	4	TPM_KEY_CONTROL	bitName	The name of the bit to be modified
7	1	4S	1	BOOL	bitValue	The value to set the bit to
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
9		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
12	20		20	TPM_AUTHDATA	ownerAuth	HMAC authorization: key ownerAuth

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal:TPM_ORD_KeyControlOwner
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM.
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	HMAC authorization: key ownerAuth

Descriptions

1. Set an internal bit within the key cache that controls some attribute of a loaded key.

Actions

1. Validate the AuthData using the owner authentication value, on error return TPM_AUTHFAIL
2. Validate that keyHandle refers to a loaded key, return TPM_INVALID_KEYHANDLE on error.
3. Validate that pubKey matches the key held by the TPM pointed to by keyHandle, return TPM_BAD_PARAMETER on mismatch
    a. This check added so that virtualization of the keyHandle does not result in attacks as the keyHandle is not associated with an authorization value
4. Validate that bitName is valid, return TPM_BAD_MODE on error.
5. If bitName == TPM_KEY_CONTROL_OWNER_EVICT
    a. If bitValue == TRUE
        i. Verify that after this operation at least two key slots will be present within the TPM that can store any type of key both of which do NOT have the OwnerEvict bit set, on error return TPM_NOSPACE

ii. Verify that for this key handle, parentPCRStatus is FALSE and isVolatile is FALSE. Return TPM_BAD_PARAMETER on error.

iii. Set ownerEvict within the internal key storage structure to TRUE.

b. Else if bitValue == FALSE

i. Set ownerEvict within the internal key storage structure to FALSE.

6. Return TPM_SUCCESS

21.2 TPM_SaveContext

Start of informative comment:

TPM_SaveContext saves a loaded resource outside the TPM. After successful execution of the command, the TPM automatically releases the internal memory for sessions but leaves keys in place.

There is no assumption that a saved context blob is stored in a safe, protected area. Since the context blob can be loaded at any time, do not rely on TPM_SaveContext to restrict access to an entity such as a key. If use of the entity should be restricted, means such as authorization secrets or PCR's should be used.

In general, TPM_SaveContext can save a transport session. However, it cannot save an exclusive transport session, because any ordinal other than TPM_ExecuteTransport terminates the exclusive transport session. This action prevents the exclusive transport session from being saved and reloaded while intervening commands are hidden from the transport log.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveContext
4	4			TPM_HANDLE	handle	Handle of the resource being saved.
5	4	2S	4	TPM_RESOURCE_TYPE	resourceType	The type of resource that is being saved
6	16	3S	16	BYTE[16]	label	Label for identification purposes

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveContext
4	4	3S	4	UINT32	contextSize	The actual size of the outgoing context blob
5	<>	4S	<>	TPM_CONTEXT_BLOB	contextBlob	The context blob

Description

1. The caller of the function uses the label field to add additional sequencing, anti-replay or other items to the blob. The information does not need to be confidential but needs to be part of the blob integrity.

Actions

1. Map V1 to TPM_STANY_DATA
2. Validate that handle points to resource that matches resourceType, return TPM_INVALID_RESOURCE on error
3. Validate that resourceType is a resource from the following list if not return TPM_INVALID_RESOURCE
    a. TPM_RT_KEY
    b. TPM_RT_AUTH
    c. TPM_RT_TRANS
    d. TPM_RT_DAA_TPM
4. Locate the correct nonce
    a. If resourceType is TPM_RT_KEY
        i. If TPM_STCLEAR_DATA -> contextNonceKey is NULLS
            (1) Set TPM_STCLEAR_DATA -> contextNonceKey to the next value from the TPM RNG
        ii. Map N1 to TPM_STCLEAR_DATA -> contextNonceKey
        iii. If the key has TPM_KEY_CONTROL_OWNER_EVICT set then return TPM_OWNER_CONTROL
    b. Else
        i. If V1 -> contextNonceSession is NULLS
            (1) Set V1 -> contextNonceSession to the next value from the TPM RNG
        ii. Map N1 to V1 -> contextNonceSession
5. Set K1 to TPM_PERMANENT_DATA -> contextKey
6. Create R1 by putting the sensitive part of the resource pointed to by handle into a structure. The structure is a TPM manufacturer option. The TPM MUST ensure that ALL sensitive information of the resource is included in R1.
7. Create C1 a TPM_CONTEXT_SENSITIVE structure
    a. C1 forms the inner encrypted wrapper for the blob. All saved context blobs MUST include a TPM_CONTEXT_SENSITIVE structure and the TPM_CONTEXT_SENSITIVE structure MUST be encrypted.
    b. Set C1 -> contextNonce to N1
    c. Set C1 -> internalData to R1
8. Create B1 a TPM_CONTEXT_BLOB
    a. Set B1 -> tag to TPM_TAG_CONTEXTBLOB
    b. Set B1 -> resourceType to resourceType
    c. Set B1 -> handle to handle
    d. Set B1 -> integrityDigest to NULL

```
    e. Set B1 -> label to label
    f. Set B1 -> additionalData to information determined by the TPM manufacturer. This data will help the TPM to reload and reset context. This area MUST NOT hold any data that is sensitive (symmetric IV are fine, prime factors of an RSA key are not).
        i. For OSAP sessions, and DSAP attached to keys, the hash of the entity MUST be included in additionalData
    g. Set B1 -> additionalSize to the size of additionalData
    h. Set B1 -> sensitiveSize to the size of C1
    i. Set B1 -> sensitiveData to C1
9. If resourceType is TPM_RT_KEY
    a. Set B1 -> contextCount to 0
10. Else
    a. If V1 -> contextCount > $2^{32}-2$ then
        i. Return with TPM_TOOMANYCONTEXTS
    b. Else
        i. Increment V1 -> contextCount by 1
        ii. Validate that the TPM can still manage the new count value
            (1) If the distance between the oldest saved context and the contextCount is too large return TPM_CONTEXT_GAP
        iii. Find contextIndex such that V1 -> contextList[contextIndex] equals 0. If not found exit with TPM_NOCONTEXTSPACE
        iv. Set V1-> contextList[contextIndex] to V1 -> contextCount
        v. Set B1 -> contextCount to V1 -> contextCount
    c. The TPM MUST invalidate all information regarding the resource except for information needed for reloading
11. Calculate B1 -> integrityDigest the HMAC of B1 using TPM_PERMANENT_DATA -> tpmProof as the secret
12. Create E1 by encrypting C1 using K1 as the key
    a. Set B1 -> sensitiveSize to the size of E1
    b. Set B1 -> sensitiveData to E1
13. Set contextSize to the size of B1
14. Return B1 in contextBlob
```

21.3 TPM_LoadContext

Start of informative comment:

TPM_LoadContext loads into the TPM a previously saved context. The command returns a handle.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadContext
4	4			TPM_HANDLE	entityHandle	The handle the TPM MUST use to locate the entity tied to the OSAP/DSAP session
5	1	2S	1	BOOL	keepHandle	Indication if the handle MUST be preserved
6	4	3S	4	UINT32	contextSize	The size of the following context blob.
7	<>	4S	<>	TPM_CONTEXT_BLOB	contextBlob	The context blob

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadContext
4	4			TPM_HANDLE	handle	The handle assigned to the resource after it has been successfully loaded.

Actions

1. Map contextBlob to B1, a TPM_CONTEXT_BLOB structure
2. Map V1 to TPM_STANY_DATA
3. Create M1 by decrypting B1 -> sensitiveData using TPM_PERMANENT_DATA -> contextKey
4. Create C1 and R1 by splitting M1 into a TPM_CONTEXT_SENSITIVE structure and internal resource data
5. Check contextNonce
    a. If B1 -> resourceType is NOT TPM_RT_KEY
        i. If C1 -> contextNonce does not equal V1 -> contextNonceSession return TPM_BADCONTEXT

ii. Validate that the resource pointed to by the context is loaded (i.e. for OSAP the key referenced is loaded and DSAP connected to the key) return TPM_RESOURCEMISSING

(1) For OSAP sessions the TPM MUST validate that the incoming pubkey hash matches the key held by the TPM

(2) For OSAP and DSAP sessions referring to a key, verify that entityHandle identifies the key linked to this OSAP/DSAP session, if not return TPM_BAD_HANDLE.

b. Else

i. If C1 -> internalData -> parentPCRStatus is FALSE and C1 -> internalData -> isVolatile is FALSE

(1) Ignore C1 -> contextNonce

ii. else

(1) If C1 -> contextNonce does not equal TPM_STCLEAR_DATA -> contextNonceKey return TPM_BADCONTEXT

6. Validate the structure

a. Set H1 to B1 -> integrityDigest

b. Set B1 -> integrityDigest to NULL

c. Copy M1 to B1 -> sensitiveData

d. Create H2 the HMAC of B1 using TPM_PERMANENT_DATA -> tpmProof as the HMAC key

e. If H2 does equal H1 return TPM_BADCONTEXT

7. If keepHandle is TRUE

a. Set handle to B1 -> handle

b. If the TPM is unable to restore the handle the TPM MUST return TPM_BAD_HANDLE

8. Else

a. The TPM SHOULD attempt to restore the handle but if not possible it MAY set the handle to any valid for B1 -> resourceType

9. If B1 -> resourceType is NOT TPM_RT_KEY

a. Find contextIndex such that V1 -> contextList[contextIndex] equals B1 -> TPM_CONTEXT_BLOB -> contextCount

b. If not found then return TPM_BADCONTEXT

c. Set V1 -> contextList[contextIndex] to 0

10. Process B1 to return the resource back into TPM use

22. Eviction

Start of informative comment:

The TPM has numerous resources held inside of the TPM that may need eviction. The need for eviction occurs when the number or resources in use by the TPM exceed the available space. For resources that are hard to reload (i.e. keys tied to PCR values) the outside entity should first perform a context save before evicting items.

In version 1.1 there were separate commands to evict separate resource types. This new command set uses the resource types defined for context saving and creates a generic command that will evict all resource types.

End of informative comment.

The TPM MUST NOT flush the EK or SRK using this command.

Version 1.2 deprecates the following commands:

? TPM_Terminate_Handle

? TPM_EvictKey

? TPM_Reset

22.1 TPM_FlushSpecific

Start of informative comment:

TPM_FlushSpecific flushes from the TPM a specific handle.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_FlushSpecific
4	4			TPM_HANDLE	handle	The handle of the item to flush
5	4	2S	4	TPM_RESOURCE_TYPE	resourceType	The type of resource that is being flushed

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_FlushSpecific

Description

TPM_FlushSpecific releases the resources associated with the given handle.

Actions

1. If resourceType is TPM_RT_CONTEXT
    a. The handle for a context is not a handle but the "context count" value. The TPM uses the "context count" value to locate the proper contextList entry and sets R1 to the contextList entry
    b. If R1 is not a valid saved context return TPM_BAD_PARAMETER
2. Else if resourceType is TPM_RT_KEY
    a. Set R1 to the key pointed to by handle
    b. Validate that R1 points at valid key
    c. If R1 -> ownerEvict is TRUE return TPM_KEY_OWNER_CONTROL
3. Else if resourceType is TPM_RT_HASH or TPM_RT_COUNTER or TPM_RT_DELEGATE
    a. Return TPM_INVALID_RESOURCE
4. Else

   a. Set R1 to the resource pointed to by handle
   b. Validate that resource type and handle point to a valid allocated resource

5. Invalidate R1 and all internal resources allocated to R1
   a. Resources include authorization sessions

23. Timing Ticks

Start of informative comment:

The TPM timing ticks are always available for use. The association of timing ticks to actual time is a protocol that occurs outside of the TPM. See the design document for details.

The setting of the clock type variable is a one time operation that allows the TPM to be configured to the type of platform that is installed on.

The ability for the TPM to continue to increment the timer ticks across power cycles of the platform is a TPM and platform manufacturer decision.

End of informative comment.

23.1 TPM_GetTicks

Start of informative comment:

This command returns the current tick count of the TPM.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetTicks

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetTicks
4	32	3S	32	TPM_CURRENT_TICKS	currentTime	The current time held in the TPM

Descriptions

This command returns the current time held in the TPM. It is the responsibility of the external system to maintain any relation between this time and a UTC value or local real time value.

Actions

1. Set T1 to the internal TPM_CURRENT_TICKS structure
2. Return T1 as currentTime.

23.2 TPM_TickStampBlob

Start of informative comment:

This command applies a time stamp to the passed blob. The TPM makes no representation regarding the blob merely that the blob was present at the TPM at the time indicated.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal, fixed value of TPM_ORD_TickStampBlob
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
5	20	2s	20	TPM_NONCE	antiReplay	Anti replay value to added to signature
6	20	3s	20	TPM_DIGEST	digestToStamp	The digest to perform the tick stamp on
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AU THDATA	privAuth	The authorization session digest that authorizes the use of keyHandle. HMAC key: key.usageAuth

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal, fixed value of TPM_ORD_TickStampBlob
4	32	3S	32	TPM_CURRENT_TICKS	currentTicks	The current time according to the TPM
5	4	4S	4	UINT32	sigSize	The length of the returned digital signature
6	<>	5S	<>	BYTE[ ]	sig	The resulting digital signature.
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
9	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

Description

The function performs a digital signature on the hash of digestToStamp and the current tick count.

It is the responsibility of the external system to maintain any relation between tick count and a UTC value or local real time value.

Actions

3. The TPM validates the AuthData to use the key pointed to by keyHandle.
4. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY or TPM_KEY_LEGACY, if not return the error code TPM_INVALID_KEYUSAGE.
5. Return TPM_INAPPROPRIATE_SIG if the keyHandle -> sigScheme is not SHA-1
6. If TPM_STCLEAR_DATA -> currentTicks is not properly initialized
    a. Initialize the TPM_STCLEAR_DATA -> currentTicks
7. Create T1, a TPM_CURRENT_TICKS structure.
8. Create H1 a TPM_SIGN_INFO structure and set the structure defaults
    a. Set H1 -> fixed to "TSTP"
    b. Set H1 -> replay to antiReplay
    c. Create H2 the concatenation of digestToStamp | | T1
    d. Set H1 -> dataLen to the length of H2
    e. Set H1 -> data to H2
9. The TPM computes the signature, sig, using the key referenced by keyHandle, using SHA-1 of H1 as the information to be signed

177 10.The TPM returns T1 as currentTicks parameter

24. Transport Sessions

24.1 TPM_EstablishTransport

Start of informative comment:

This establishes the transport session. Depending on the attributes specified for the session this may establish shared secrets, encryption keys, and session logs. The session will be in use for by the TPM_ExecuteTransport command.

The only restriction on what can happen inside of a transport session is that there is no "nesting" of sessions. It is permissible to perform operations that delete internal state and make the TPM inoperable.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_EstablishTransport
4	4			TPM_KEY_HANDLE	encHandle	The handle to the key that encrypted the blob
5	<>	2S	<>	TPM_TRANSPORT_PUBLIC	transPublic	The public information describing the transport session
6	4	3S	4	UINT32	secretSize	The size of the secret Area
7	<>	4S	<>	BYTE[]	secret	The encrypted secret area
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue us e flag for the authorization session handle
11	20			TPM_AUTHDATA	keyAuth	Authorization. HMAC key: encKey.usageAuth

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_EstablishTransport
4	4			TPM_TRANSHANDLE	transHandle	The handle for the transport session
5	4	3S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
6	32	4S	32	TPM_CURRENT_TICKS	currentTicks	The current tick count
7	20	5S	20	TPM_NONCE	transNonceEven	The even nonce in use for subsequent execute transport
8	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key: key.usageAuth

Description

This command establishes the transport sessions shared secret. The encryption of the shared secret uses the public key of the key loaded in encKey.

Actions

1. If encHandle is TPM_KH_TRANSPORT then
   a. If tag is NOT TPM_TAG_RQU_COMMAND return TPM_BADTAG
   b. If transPublic -> transAttributes specifies TPM_TRANSPORT_ENCRYPT return TPM_BAD_SCHEME
   c. If secretSize is not 20 return TPM_BAD_PARAM_SIZE
   d. Set A1 to secret
2. Else
   a. encHandle -> keyUsage MUST be TPM_KEY_STORAGE or TPM_KEY_LEGACY return TPM_INVALID_KEYUSAGE on error
   b. If encHandle -> authDataUsage does not equal TPM_AUTH_NEVER and tag is NOT TPM_TAG_RQU_AUTH1_COMMAND return TPM_AUTHFAIL
   c. Using encHandle -> usageAuth validate the AuthData to use the key and the parameters to the command
   d. Create K1 a TPM_TRANSPORT_AUTH structure by decrypting secret using the key pointed to by encHandle
   e. Validate K1 for tag
   f. Set A1 to K1 -> authData
3. If transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT

a. If TPM_PERMANENT_FLAGS -> FIPS is true and transPublic -> algId is equal to TPM_ALG_MGF1 return TPM_INAPPROPRIATE_ENC

b. Check if the transPublic -> algId is supported, if not return TPM_BAD_KEY_PROPERTY

c. If transPublic -> algid is TPM_ALG_3DES or TPM_ALG_AESXXX, check that transPublic -> encScheme is supported, if not return TPM_INAPPROPRIATE_ENC

d. Perform any initializations necessary for the algorithm

4. Generate transNonceEven from the TPM RNG

5. Create T1 a TPM_TRANSPORT_INTERNAL structure

    a. Ensure that the TPM has sufficient internal space to allocate the transport session, return TPM_RESOURCES on error

    b. Assign a T1 -> transHandle value. This value is assigned by the TPM

    c. Set T1 -> transDigest to NULL

    d. Set T1 -> transPublic to transPublic

    e. Set T1-> transNonceEven to transNonceEven

    f. Set T1 -> authData to A1

6. If TPM_STANY_DATA -> currentTicks is not properly initialized

    a. Initialize the TPM_STANY_DATA -> currentTicks

7. Set currentTicks to TPM_STANY_DATA -> currentTicks

8. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then

    a. Create L1 a TPM_TRANSPORT_LOG_IN structure

        i. Set L1 -> parameters to SHA-1 (ordinal || transPublic || secretSize || secret)

        ii. Set L1 -> pubKeyHash to NULL

        iii. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L1)

    b. Create L2 a TPM_TRANSPORT_LOG_OUT structure

        i. Set L2 -> parameters to SHA-1 (returnCode || ordinal || locality || currentTicks || transNonceEven)

        ii. Set L2 -> locality to the locality of this command

        iii. Set L2 -> currentTicks to currentTicks, this MUST be the same value that is returned in the currentTicks parameter

        iv. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L2)

9. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_EXCLUSIVE then set TPM_STANY_FLAGS -> transportExclusive to TRUE

    a. Execution of any command other than TPM_ExecuteTransport or TPM_ReleaseTransportSigned targeting this transport session will cause the abnormal invalidation of this transport session transHandle

b. The TPM gives no indication, other than invalidation of transHandle, that the session is terminated

10. Return T1 -> transHandle as transHandle

24.2 TPM_ExecuteTransport

Start of informative comment:

Delivers a wrapped TPM command to the TPM where the TPM unwraps the command and then executes the command.

TPM_ExecuteTransport uses the same rolling nonce paradigm as other authorized TPM commands. The even nonces start in TPM_EstablishTransport and change on each invocation of TPM_ExecuteTransport.

The only restriction on what can happen inside of a transport session is that there is no "nesting" of sessions. It is permissible to perform operations that delete internal state and make the TPM inoperable.

Because, in general, key handles are not logged, a digest of the corresponding public key is logged. In cases where the key handle is logged (e.g. TPM_OwnerReadInternalPub), the public key is also logged.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ExecuteTransport
4	4	2S	4	UINT32	wrappedCmdSize	Size of the wrapped command
5	<>	3S	<>	BYTE[]	wrappedCmd	The wrapped command
6	4			TPM_TRANSHANDLE	transHandle	The transport session handle
		2H1	20	TPM_NONCE	transLastNonceEven	Even nonce previously generated by TPM
7	20	3H1	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
8	1	4H1	1	BOOL	continueTransSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	transAuth	HMAC for transHandle key: transHandle -> authData

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the ExecuteTransport command. This does not reflect the status of wrapped command.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ExecuteTransport
4	8	3S	8	UINT64	currentTicks	The current ticks when the command was executed
5	4	4S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
6	4	5S	4	UINT32	wrappedRspSize	Size of the wrapped response
7	<>	6S	<>	BYTE[]	wrappedRsp	The wrapped response
8	20	2H1	20	TPM_NONCE	transNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueTransSession	The continue use flag for the session
10	20			TPM_AUTHDATA	transAuth	HMAC for transHandle key: transHandle -> authData

Description

1. This command executes a TPM command using the transport session.
2. Prior to execution of the wrapped command (action 11 below) failure of the transport session MUST have no effect on the resources referenced by the wrapped command. The exception is when the TPM goes into failure mode and return FAILED_SELFTEST for all subsequent commands.
3. After execution of the wrapped command, failure of the transport session MUST have an effect on the wrapped command resources. The reason for this is that the transport session will be returning an error code and not reporting any session nonces. The entire wrapped command response is lost so nonces, handles and such are lost to the caller.
4. Execution of the wrapped command (action 11) SHOULD have no effect on the transport session.
    a. The wrapped command SHALL use no resources of the transport session, this includes authorization sessions
    b. If the wrapped command execution returns an error (action 11 below) then the sessions for TPM_ExecuteTransport still operate properly.
    c. The exception to this is when the wrapped command causes the TPM to go into failure mode and return TPM_FAILSELFTEST for all subsequent commands
5. Field layout
    a. Command representation
    b. **
    c. TAGet | LENet | ORDet | wrappedCmdSize | wrappedCmd | AUTHet
    d. **

e. wrappedCmd looks like the following

f. **

g. TAGw | LENw | ORDw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

h. **

i. | LEN1 |

j. | E1 | (encrypted)

k. | C1 | (decrypted)

l. Response representation

m. ***

n. TAGet | LENet | RCet | wrappedRspSize | wrappedRsp | AUTHet

o. ***

p. wrappedRsp looks like the following

q. **

r. TAGw | LENw | RCw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

s. **

t. | LEN2 |

u. | ←------------------------- C2 --→ |

v. | S2 | (decrypted)

w. | E2 | (encrypted)

x. The only parameter that is possibly encrypted is DATAw

6. Additional DATAw comments
   a. For TPM_FlushSpecific and TPM_SaveContext
      i. The DATAw part of these commands does not include the handle.
         (1) It is understood that encrypting the resourceType prevents a determination of the handle type.
      ii. If the resourceType is TPM_RT_KEY, then the public key SHOULD be logged.
   b. For TPM_DAA_Join and TPM_DAA_Sign
      i. The DATAw part of these commands does not include the handle
   c. For TPM_LoadKey2
      i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or logged by the outgoing transport.
   d. For TPM_LoadKey
      i. The outgoing handle is part of the outgoing DATAw and is encrypted.
   e. For TPM_LoadContext

i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or logged by the outgoing transport.

(1) It is understood that encrypting the contextBlob prevents a determination of the handle type.

7. TPM_ExecuteTransport returns an implementation defined result when the wrapped command would cause termination of the transport session. Implementation defined possibilities include but are not limited to: the wrapped command may execute, completely, partially, or not at all, the transport session may or not be terminated, continueTransSession may not be processed or returned correctly, and an error may or may not be returned. The wrapped commands include:

   a. TPM_FlushSpecific, TPM_SaveContext targeting the transport session

   b. TPM_OwnerClear, TPM_ForceClear, TPM_RevokeTrust

Actions

1. Using transHandle locate the TPM_TRANSPORT_INTERNAL structure T1

2. Parse wrappedCmd

   a. Set TAGw, LENw, and ORDw to the parameters from wrappedCmd

   b. Set E1 to DATAw

      i. This pointer is ordinal dependent and requires the execute transport command to parse wrappedCmd

   c. Set LEN1 to the length of DATAw

      i. DATAw always ends at the start of AUTH1w if AUTH1w is present

3. If LEN1 is less that 0, or if ORDw is unknown, unimplemented, or cannot be determined

   a. Return TPM_BAD_PARAMETER

4. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT set then

   a. If T1 -> transPublic -> algId is TPM_ALG_MGF1

      i. Using the MGF1 function, create string G1 of length LEN1. The inputs to the MGF1 are transLastNonceEven, transNonceOdd, "in", and T1 -> authData. These four values concatenated together form the Z value that is the seed for the MGF1.

      ii. Create C1 by performing an XOR of G1 and wrappedCmd starting at E1.

   b. If the encryption algorithm requires an IV calculate the IV values

      i. Using the MGF1 function, create string IV1 with a length set by the block size of the encryption algorithm. The inputs to the MGF1 are transLastNonceEven, transNonceOdd, and "in". These three values concatenated together form the Z value that is the seed for the MGF1. Note that any terminating characters within the string "in" are ignored, so a total of 42 bytes are hashed.

      ii. Blocksize for TPM_ALG_DES is 8

      iii. Blocksize for TPM_ALG_AESxxx is 16

      iv. The symmetric key is taken from the first bytes of T1 -> authData.

v. Decrypt DATAw and replace the DATAw area of E1 creating C1

c. TPM_OSAP, TPM_OIAP have no parameters encrypted

d. TPM_DSAP has special rules for parameter encryption

5. Else

a. Set C1 to the DATAw area E1 of wrappedCmd

6. Create H1 the SHA-1 of (ORDw || C1).

a. C1 MUST point at the decrypted DATAw area of E1

b. The TPM MAY use this calculation for both execute transport authorization, authorization of the wrapped command and transport log creation

7. Validate the incoming transport session authorization

a. Set inParamDigest to SHA-1 (ORDet || wrappedCmdSize || H1)

b. Calculate the HMAC of (inParamDigest || transLastNonceEven || transNonceOdd || continueTransSession) using T1 -> authData as the HMAC key

c. Validate transAuth, on errors return TPM_AUTHFAIL

8. If TPM_ExecuteTransport requires auditing

a. Create TPM_AUDIT_EVENT_IN using H1 as the input parameter digest and update auditDigest

b. On any error return TPM_AUDITFAIL_UNSUCCESSFUL

9. If ORDw is from the list of fdlowing commands return TPM_NO_WRAP_TRANSPORT

a. TPM_EstablishTransport

b. TPM_ExecuteTransport

c. TPM_ReleaseTransportSigned

10. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then

a. Create L2 a TPM_TRANSPORT_LOG_IN structure

b. Set L2 -> parameters to H1

c. If ORDw is a command with no key handles

i. Set L2 -> pubKeyHash to NULL

d. If ORDw is a command with one key handle

i. Create K2 the hash of the TPM_STORE_PUBKEY structure of the key pointed to by the key handle.

ii. Set L2 -> pubKeyHash to SHA-1 (K2)

e. If ORDw is a command with two key handles

i. Create K2 the hash of the TPM_STORE_PUBKEY structure of the key pointed to by the first key handle.

ii. Create K3 the hash of the TPM_STORE_PUBKEY structure of the key pointed to by the second key handle.

        iii. Set L2 -> pubKeyHash to SHA-1 (K2 || K3)

    f. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L2)

    g. If ORDw is a command with key handles, and the key is not loaded, return TPM_INVALID_KEYHANDLE.

11. Send the wrapped command to the normal TPM command parser, the output is C2 and the return code is RCw

    a. If ORDw is a command that is audited then the TPM MUST perform the input and output audit of the command as part of this action.

    b. The TPM MAY use H1 as the data value in the authorization and audit calculations during the execution of C1

12. Set CT1 to TPM_STANY_DATA -> currentTicks -> currentTicks and return CT1 in the currentTicks output parameter

13. Calculate S2 the pointer to the DATAw area of C2

    a. Calculate LEN2 the length of S2 according to the same rules that calculated LEN1

14. Create H2 the SHA-1 of (RCw || ORDw || S2)

    a. The TPM MAY use this calculation for execute transport authorization and transport log out creation

15. Calculate the outgoing transport session authorization

    a. Create the new transNonceEven for the output of the command

    b. Set outParamDigest to SHA-1 (RCet || ORDet || TPM_STANY_DATA -> currentTicks -> currentTicks || locality || wrappedRspSize || H2)

    c. Calculate transAuth, the HMAC of (outParamDigest || transNonceEven || transNonceOdd || continueTransSession) using T1 -> authData as the HMAC key

16. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then

    a. Create L3 a TPM_TRANSPORT_LOG_OUT structure

    b. Set L3 -> parameters to H2

    c. Set L3 -> currentTicks to TPM_STANY_DATA -> currentTicks

    d. Set L3 -> locality to TPM_STANY_DATA -> localityModifier

    e. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L3)

17. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT set then

    a. If T1 -> transPublic -> AlgId is TPM_ALG_MGF1

        i. Using the MGF1 function, create string G2 of length LEN2. The inputs to the MGF1 are transNonceEven, transNonceOdd, "out", and T1 -> authData. These four values concatenated together form the Z value that is the seed for the MGF1.

        ii. Create E2 by performing an XOR of G2 and C2 starting at S2.

    b. Else

    i. Create IV2 using the same algorithm as IV1 with the input values transNonceEven, transNonceOdd, and "out". Note that any terminating characters within the string "out" are ignored, so a total of 43 bytes are hashed.

    ii. Create E2 by encrypting C2 starting at S2 using IV2

18. Else

    a. Set E2 to the DATAw area S2 of wrappedRsp

19. If continueTransSession is FALSE

    a. Invalidate all session data related to transHandle

20. If TPM_ExecuteTranport requires auditing

    a. Create TPM_AUDIT_EVENT_OUT using H2 for the parameters and update the auditDigest

    b. On any errors return TPM_AUDITFAIL_SUCCESSFUL or TPM_AUDITFAIL_UNSUCCESSFUL depending on RCw

21. Return C2 but with S2 replaced by E2 in the wrappedRsp parameter

24.3 TPM_ReleaseTransportSigned

Start of informative comment:

This command completes the transport session. If logging for this session is turned on, then this command returns a hash of all operations performed during the session along with a digital signature of the hash.

This command serves no purpose if logging is turned off, and results in an error if attempted.

This command uses two authorization sessions, the key that will sign the log and the authorization from the session. Having the session authorization proves that the requestor that is signing the log is the owner of the session. If this restriction is not put in then an attacker can close the log and sign using their own key.

The hash of the session log includes the information associated with the input phase of execution of the TPM_ReleaseTransportSigned command. It cannot include the output phase information.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseTransportSigned
4	4			TPM_KEY_HANDLE	keyHandle	Handle of a loaded key that will perform the signing
5	20	2S	20	TPM_NONCE	antiReplay	Value provided by caller for anti-replay protection
6	4			TPM_AUTHHANDLE	authHandle	The authorization session to use key
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	keyAuth	The authorization session digest that authorizes the use of key. HMAC key: key -> usageAuth
10	4			TPM_TRANSHANDLE	transHandle	The transport session handle
		2H2	20	TPM_NONCE	transLastNonceEven	Even nonce in use by execute Transport
11	20	3H2	20	TPM_NONCE	transNonceOdd	Nonce supplied by caller for transport session
12	1	4H2	1	BOOL	continueTransSession	The continue use flag for the authorization session handle
13	20			TPM_AUTHDATA	transAuth	HMAC for transport session key: transHandle -> authData

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseTransportSigned
4	4	3S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
5	32	4S	32	TPM_CURRENT_TICKS	currentTicks	The current ticks when the command executed
6	4	5S	4	UINT32	signSize	The size of the signature area
7	<>	6S	<>	BYTE[]	signature	The signature of the digest
8	20	2H1	20	TPM_NONCE	authNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the session
10	20			TPM_AUTHDATA	keyAuth	HMAC: key -> usageAuth
11	20	2H2	20	TPM_NONCE	transNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
12	1	4H2	1	BOOL	continueTransSession	The continue use flag for the session
13	20			TPM_AUTHDATA	transAuth	HMAC: transHandle -> authData

Description

This command releases a transport session and signs the transport log

Actions

1. Using transHandle locate the TPM_TRANSPORT_INTERNAL structure T1
2. Return TPM_INAPPROPRIATE_SIG if the key -> sigScheme is not SHA-1
3. Using key -> authData validate the command and parameters, on error return TPM_AUTHFAIL
4. Using transHandle -> authData validate the command and parameters, on error return TPM_AUTH2FAIL
5. If T1 -> transAttributes has TPM_TRANSPORT_LOG set then
   a. Create A1 a TPM_TRANSPORT_LOG_OUT structure
   b. Set A1 –> parameters to the SHA-1 (ordinal || antiReplay)
   c. Set A1 -> currentTicks to TPM_STANY_DATA -> currentTicks
   d. Set A1 -> locality to the locality modifier for this command
   e. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || A1)
6. Else
   a. Return TPM_BAD_MODE

7. Create H1 a TPM_SIGN_INFO structure and set the structure defaults
    a. Set H1 -> fixed to "TRAN"
    b. Set H1 -> replay to antiReplay
    c. Set H1 -> data to T1 -> transDigest
    d. Sign SHA-1 hash of H1 using the key pointed to by key
8. Invalidate all session data related to T1
9. Set continueTransSession to FALSE
10. Return TPM_SUCCESS

25. Monotonic Counter

25.1 TPM_CreateCounter

Start of informative comment:

This command creates the counter but does not select the counter. Counter creation assigns an AuthData value to the counter and sets the counters original start value. The original start value is the current internal base value plus one. Setting the new counter to the internal base avoids attacks on the system that are attempting to use old counter values.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateCounter
4	20	2S	20	TPM_ENCAUTH	encAuth	The encrypted auth data for the new counter
5	4	3s	4	BYTE	label	Label to associate with counter
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Ignored
10	20		20	TPM_AUTHDATA	ownerAuth	Authorization ow nerAuth.

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateCounter
4	4	3s	4	TPM_COUNT_ID	countID	The handle for the counter
5	10	4S	10	TPM_COUNTER_VALUE	counterValue	The starting counter value
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Fixed value of FALSE
8	20		20	TPM_AUTHDATA	resAuth	Authorization. HMAC key: ownerAuth.

Description

This command creates a new monotonic counter. The TPM MUST support a minimum of 4 concurrent counters.

Actions

The TPM SHALL do the following:

1. Using the authHandle field, validate the owner's AuthData to execute the command and all of the incoming parameters. The authorization session MUST be OSAP or DSAP
2. Ignore continueAuthSession on input and set continueAuthSession to FALSE on output
3. If authHandle indicates XOR encryption for the AuthData secrets
   a. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
   b. Create a1 by XOR X1 and encAuth
4. Else
   a. Create a1 by decrypting encAuth using the algorithm indicated in the OSAP session
   b. Key is from authHandle -> sharedSecret
   c. IV is SHA-1 of (authLastNonceEven || nonceOdd)
5. Validate that there is sufficient internal space in the TPM to create a new counter. If there is insufficient space, the command returns an error.
   a. The TPM MUST provide storage for a1, TPM_COUNTER_VALUE, countID, and any other internal data the TPM needs to associate with the counter
6. Increment the max counter value
7. Set the counter to the max counter value
8. Set the counter label to label

526 9. Create a countID

25.2 TPM_IncrementCounter

Start of informative comment:

This authorized command increments the indicated counter by one. Once a counter has been incremented then all subsequent increments must be for the same handle until a successful TPM_Startup(ST_CLEAR) is executed.

The order for checking validation of the command parameters when no counter is active, keeps an attacker from creating a denial-of-service attack.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_IncrementCounter
4	4	2s	4	TPM_COUNT_ID	countID	The handle of a valid counter
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for counter authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	counterAuth	The authorization session digest that authorizes the use of countID. HMAC key: countID -> authData

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_IncrementCounter
5	10	3S	10	TPM_COUNTER_VALUE	count	The counter value
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: countID -> authData

Description

This function increments the counter by 1.

The TPM MAY implement increment throttling to avoid burn problems

Actions

1. If TPM_STCLEAR_DATA -> countID is NULL
    a. Validate that countID is a valid counter, return TPM_BAD_COUNTER on mismatch
    b. Validate the command parameters using counterAuth
    c. Set TPM_STCLEAR_DATA -> countID to countID
2. else
    a. If TPM_STCLEAR_DATA -> countID does not equal countID
        i. Return TPM_BAD_COUNTER
    b. Validate the command parameters using counterAuth
3. Increments the counter by 1
4. Return new count value in count

25.3 TPM_ReadCounter

Start of informative comment:

Reading the counter provides the caller with the current number in the sequence.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadCounter
4	4	2S	4	TPM_COUNT_ID	countID	ID value of the counter

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadCounter
4	10	3S	4	TPM_COUNTER_VALUE	count	The counter value

Description

This returns the current value for the counter indicated. The counter MAY be any valid counter.

Actions

1. Validate that countID points to a valid counter. Return TPM_BAD_COUNTER on error.
2. Return count

25.4 TPM_ReleaseCounter

Start of informative comment:

This command releases a counter such that no reads or increments of the indicated counter will succeed.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseCounter
4	4	2s	4	TPM_COUNT_ID	countID	ID value of the counter
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for countID authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce associated with countID
7	1	4H1	1	BOOL	continueAuthSession	Ignored
8	20			TPM_AUTHDATA	counterAuth	The authorization session digest that authorizes the use of countID. HMAC key: countID -> authData

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseCounter
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: countID -> authData

Actions

The TPM uses countID to locate a valid counter.

1. Authenticate the command and the parameters using the AuthData pointed to by countID. Return TPM_AUTHFAIL on error
2. The TPM invalidates all internal information regarding the counter. This includes releasing countID such that any subsequent attempts to use countID will fail.
3. The TPM invalidates sessions

   a. MUST invalidate all OSAP sessions associated with the counter
   b. MAY invalidate any other session
4. If TPM_STCLEAR_DATA -> countID equals countID,
   a. Set TPM_STCLEAR_DATA -> countID to an illegal value (not the NULL value)

25.5 TPM_ReleaseCounterOwner

Start of informative comment:

This command releases a counter such that no reads or increments of the indicated counter will succeed.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseCounterOwner
4	4	2s	4	TPM_COUNT_ID	countID	ID value of the counter
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest that authorizes the inputs. HMAC key: ownerAuth

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseCounterOwner
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth

Description

This invalidates all information regarding a counter.

Actions

1. Validate that ownerAuth properly authorizes the command and parameters
2. The TPM uses countID to locate a valid counter. Return TPM_BAD_COUNTER if not found.

3. The TPM invalidates all internal information regarding the counter. This includes releasing countID such that any subsequent attempts to use countID will fail.
4. The TPM invalidates sessions
    a. MUST invalidate all OSAP sessions associated with the counter
    b. MAY invalidate any other session
5. If TPM_STCLEAR_DATA -> countID equals countID,
    a. Set TPM_STCLEAR_DATA -> countID to an illegal value (not the NULL value)

26. DAA commands

26.1 TPM_DAA_Join

Start of informative comment:

TPM_DAA_Join is the process that establishes the DAA parameters in the TPM for a specific DAA issuing authority.

End of informative comment.

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DAA_Join.
4	4			TPM_HANDLE	handle	Session handle
5	1	2S	1	BYTE	stage	Processing stage of join
6	4	3S	4	UINT32	inputSize0	Size of inputData0 for this stage of JOIN
7	<>	4S	<>	BYTE[]	inputData0	Data to be used by this capability
8	4	5S	4	UINT32	inputSize1	Size of inputData1 for this stage of JOIN
9	<>	6S	<>	BYTE[]	inputData1	Data to be used by this capability
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2 H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
13	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

608 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes incl. paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DAA_Join.
4	4	3S	4	UINT32	outputSize	Size of outputData
5	<>	4S	<>	BYTE[]	outputData	Data produced by this capability
6	20	2 H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20		20	TPM_AUTHDATA	resAuth	Authorization HMAC key: ownerAuth.

Description

This table summaries the input, output and saved data that is associated with each stage of processing.

Stage	Input Data0	Input Data1	Operation	Output Data	Scratchpad
0	DAA_count (used as # repetitions of stage 1)	NULL	initialise	Session Handle	NULL
1	n0	signatureValue	rekeying	NULL	n0
2	DAA_issuerSettings	signatureValue	issuer settings	NULL	NULL
3	DAA_count	NULL	DAA_join_uo, DAA_join_u1	NULL	NULL
4	DAA_generic_R0	DAA_generic_n	P1=R0^f0 mod n	NULL	P1
5	DAA_generic_R1	DAA_generic_n	P2 = P1*(R1^f1) mod n	NULL	P2
6	DAA_generic_S0	DAA_generic_n	P3 = P2*(S0^u0) mod n	NULL	P3
7	DAA_generic_S1	DAA_generic_n	U = P3*(S1^u1) mod n	U	NULL
8	NE	NULL	U2	U2	NULL
9	DAA_generic_R0	DAA_generic_n	P1=R0^r0 mod n	NULL	P1
10	DAA_generic_R1	DAA_generic_n	P2 = P1*(R1^r1) mod n	NULL	P2
11	DAA_generic_S0	DAA_generic_n	P3 = P2*(S0^r2) mod n	NULL	P3
12	DAA_generic_S1	DAA_generic_n	P4 = P3*(S1^r3) mod n	P4	NULL
13	DAA_generic_gamma	w	w1 = w^q mod gamma	NULL	w
14	DAA_generic_gamma	NULL	E = w^f mod gamma	E	w
15	DAA_generic_gamma	NULL	r = r0 + (2^power0)*r1 mod q, E1 = w^r mod gamma	E1	NULL
16	c1	NULL	c = hash(c1 || NT)	nt	NULL
17	NULL	NULL	s0 = r0 + c*f0	s0	NULL
18	NULL	NULL	s1 = r1 + c*f1	s1	NULL
19	NULL	NULL	s2 = r2 + c*u0 mod 2^power1	s2	NULL
20	NULL	NULL	s12 = r2 + c*u0 >> power1	c	s12
21	NULL	NULL	s3 = r3 + c*u1 + s12	s3	NULL
22	u2	NULL	v0 = u2 + u0 mod 2^power1 v10 = u2 + u0 >> power1	enc(v0)	v10
23	u3	NULL	V1 = u3 + u1 + v10	enc(v1)	NULL
24	NULL	NULL	enc(DAA_tpmSpecific)	enc(DAA_tpmSpecific)	NULL

Actions

A Trusted Platform Module that receives a valid TPM_DAA_Join command SHALL:

1. Use ownerAuth to verify that the Owner authorized all TPM_DAA_Join input parameters.
2. Any error return results in the TPM invalidating all resources associated with the join
3. Constant values of 0 or 1 are 1 byte integers, stages affected are
   a. 4(j), 5(j), 14(f), 17(e)
4. Representation of the strings "r0" to "r4" are 2-byte ASCII encodings, stages affected are
   a. 9(i), 10(h), 11(h), 12(h), 15(f),15(g), 17(d), 18(d), 19(d), 20(d), 21(d)

Stages

0. If stage==0
   a. Determine that sufficient resources are available to perform a TPM_DAA_Join.
      i. The TPM MUST support sufficient resources to perform one (1) TPM_DAA_Join/ TPM_DAA_Sign. The TPM MAY support additional TPM_DAA_Join/ TPM_DAA_Sign sessions.
      ii. The TPM may share internal resources between the DAA operations and other variable resource requirements:
         iii. If there are insufficient resources within the stored key pool (and one or more keys need to be removed to permit the DAA operation to execute) return TPM_NOSPACE
         iv. If there are insufficient resources within the stored session pool (and one or more authorization or transport sessions need to be removed to permit the DAA operation to execute), return TPM_RESOURCES.
   b. Set all fields in DAA_issuerSettings = NULL
   c. set all fields in DAA_tpmSpecific = NULL
   d. set all fields in DAA_session = NULL
   e. Set all fields in DAA_JoinSession = NULL
   f. Verify that sizeOf(inputData0) == sizeOf(DAA_tpmSpecific -> DAA_count) and return error TPM_DAA_INPUT_DATA0 on mismatch
   g. Verify that inputData0 > 0, and return error TPM_DAA_INPUT_DATA0 on mismatch
   h. Set DAA_tpmSpecific -> DAA_count = inputData0
   i. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession))
   j. set DAA_session -> DAA_stage = 1
   k. Assign session handle for TPM_DAA_Join
   l. set outputData = new session handle
   m. return TPM_SUCCESS

```
1. If stage==1
   a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return TPM_DAA_TPM_SETTINGS on mismatch
   c. Verify that sizeOf(inputData0) == DAA_SIZE_issuerModulus and return error TPM_DAA_INPUT_DATA0 on mismatch
   d. If DAA_session -> DAA_scratch == NULL:
      i. Set DAA_session -> DAA_scratch = inputData0
      ii. set DAA_joinSession -> DAA_digest_n0 = SHA1(DAA_session -> DAA_scratch)
      iii. set DAA_tpmSpecific -> DAA_rekey = SHA1(TPM_DAA_TPM_SEED || DAA_joinSession -> DAA_digest_n0)
   e. Else (If DAA_session -> DAA_scratch != NULL):
      i. Set signedData = inputData0
      ii. Verify that sizeOf(inputData1) == DAA_SIZE_issuerModulus and return error TPM_DAA_INPUT_DATA1 on mismatch
      iii. Set signatureValue = inputData1
      iv. Use the RSA key == [DAA_session -> DAA_scratch] to verify that signatureValue is a signature on signedData, and return error TPM_DAA_ISSUER_VALIDITY on mismatch
      v. Set DAA_session -> DAA_scratch = signedData
   f. Decrement DAA_tpmSpecific -> DAA_count by 1 (unity)
   g. If DAA_tpmSpecific -> DAA_count ==0:
   h. increment DAA_Session -> DAA_Stage by 1
   i. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession)
   j. set outputData = NULL
   k. return TPM_SUCCESS
2. If stage==2
   a. Verify that DAA_session ->DAA_stage==2. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
   c. Verify that sizeOf(inputData0) == sizeOf(TPM_DAA_ISSUER) and return error TPM_DAA_INPUT_DATA0 on mismatch
   d. Set DAA_issuerSettings = inputData0. Verify that all fields in DAA_issuerSettings are present and return error TPM_DAA_INPUT_DATA0 if not.
```

e. Verify that sizeOf(inputData1) == DAA_SIZE_issuerModulus and return error TPM_DAA_INPUT_DATA1 on mismatch

f. Set signatureValue = inputData1

g. Set signedData = (DAA_joinSession -> DAA_digest_n0 | | DAA_issuerSettings)

h. Use the RSA key [DAA_session -> DAA_scratch] to verify that signatureValue is a signature on signedData, and return error TPM_DAA_ISSUER_VALIDITY on mismatch

i. Set DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)

j. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific | | DAA_joinSession)

k. Set DAA_session -> DAA_scratch = NULL

l. increment DAA_session -> DAA_stage by 1

m. return TPM_SUCCESS

3. If stage==3

a. Verify that DAA_session ->DAA_stage==3. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpedific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific | | DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Verify that sizeOf(inputData0) == sizeOf(DAA_tpmSpecific -> DAA_count) and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set DAA_tpmSpecific -> DAA_count = inputData0

f. obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u0

g. obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u1

h. set outputData = NULL

i. increment DAA_session -> DAA_stage by 1

j. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific | | DAA_joinSession)

k. return TPM_SUCCESS

4. If stage==4,

a. Verify that DAA_session ->DAA_stage==4. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific | | DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_R0 = inputData0

e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch

h. Set X = DAA_generic_R0

i. Set n = DAA_generic_n

j. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod DAA_issuerSettings -> DAA_generic_q

k. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)

l. Set DAA_session -> DAA_scratch = (X^f0) mod n

m. set outputData = NULL

n. increment DAA_session -> DAA_stage by 1

o. return TPM_SUCCESS

5. If stage==5

a. Verify that DAA_session ->DAA_stage==5. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_R1 = inputData0

e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and return error TPM_DAA_INPUT_DATA0on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch

h. Set X = DAA_generic_R1

i. Set n = DAA_generic_n

j. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod DAA_issuerSettings -> DAA_generic_q.

k. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the result f1

l. Set Z = DAA_session -> DAA_scratch

m. Set DAA_session -> DAA_scratch = Z*(X^f1) mod n

n. set outputData = NULL

   o. increment DAA_session -> DAA_stage by 1

   p. return TPM_SUCCESS

6. If stage==6

   a. Verify that DAA_session ->DAA_stage==6. Return TPM_DAA_STAGE and flush handle on mismatch

   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

   d. Set DAA_generic_S0 = inputData0

   e. Verify that SHA-1(DAA_generic_S0) == DAA_issuerSettings -> DAA_digest_S0 and return error TPM_DAA_INPUT_DATA0 on mismatch

   f. Set DAA_generic_n = inputData1

   g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch

   h. Set X = DAA_generic_S0

   i. Set n = DAA_generic_n

   j. Set Z = DAA_session -> DAA_scratch

   k. Set Y = DAA_joinSession -> DAA_join_u0

   l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

   m. set outputData = NULL

   n. increment DAA_session -> DAA_stage by 1

   o. return TPM_SUCCESS

7. If stage==7

   a. Verify that DAA_session ->DAA_stage==7. Return TPM_DAA_STAGE and flush handle on mismatch

   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

   d. Set DAA_generic_S1 = inputData0

   e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and return error TPM_DAA_INPUT_DATA0 on mismatch

   f. Set DAA_generic_n = inputData1

   g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch

   h. Set X = DAA_generic_S1

i. Set n = DAA_generic_n

j. Set Y = DAA_joinSession -> DAA_join_u1

k. Set Z = DAA_session -> DAA_scratch

l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

m. Set DAA_session -> DAA_digest to the SHA-1 (DAA_session -> DAA_scratch || DAA_tpmSpecific -> DAA_count || DAA_joinSession -> DAA_digest_n0)

n. set outputData = DAA_session -> DAA_scratch

o. set DAA_session -> DAA_scratch = NULL

p. increment DAA_session -> DAA_stage by 1

q. return TPM_SUCCESS

8. If stage==8

a. Verify that DAA_session ->DAA_stage==8. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Verify inputSize0 == DAA_SIZE_NE and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set NE = decrypt(inputData0, privEK)

f. set outputData = SHA-1(DAA_session -> DAA_digest || NE)

g. set DAA_session -> DAA_digest = NULL

h. increment DAA_session -> DAA_stage by 1

i. return TPM_SUCCESS

9. If stage==9

a. Verify that DAA_session ->DAA_stage==9. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_R0 = inputData0

e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch

   h. obtain random data from the RNG and store it as DAA_session -> DAA_contextSeed

   i. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them Y

   j. Set X = DAA_generic_R0

   k. Set n = DAA_generic_n

   l. Set DAA_session -> DAA_scratch = (X^Y) mod n

   m. set outputData = NULL

   n. increment DAA_session -> DAA_stage by 1

   o. return TPM_SUCCESS

10. If stage==10

   a. Verify that DAA_session ->DAA_stage==10. Return TPM_DAA_STAGE and flush handle on mismatch h

   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

   d. Set DAA_generic_R1 = inputData0

   e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and return error TPM_DAA_INPUT_DATA0 on mismatch

   f. Set DAA_generic_n = inputData1

   g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch

   h. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them Y

   i. Set X = DAA_generic_R1

   j. Set n = DAA_generic_n

   k. Set Z = DAA_session -> DAA_scratch

   l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

   m. set outputData = NULL

   n. increment DAA_session -> DAA_stage by 1

   o. return TPM_SUCCESS

11. If stage==11

   a. Verify that DAA_session ->DAA_stage==11. Return TPM_DAA_STAGE and flush handle on mismatch

   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_S0 = inputData0

e. Verify that SHA-1(DAA_generic_S0) == DAA_issuerSettings -> DAA_digest_S0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch

h. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them Y

i. Set X = DAA_generic_S0

j. Set n = DAA_generic_n

k. Set Z = DAA_session -> DAA_scratch

l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

m. set outputData = NULL

n. increment DAA_session -> DAA_stage by 1

o. return TPM_SUCCESS

12. If stage==12

a. Verify that DAA_session ->DAA_stage==12. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_S1 = inputData0

e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch

h. obtain DAA_SIZE_r3 bits from MGF1("r3", DAA_session -> DAA_contextSeed), and label them Y

i. Set X = DAA_generic_S1

j. Set n = DAA_generic_n

k. Set Z = DAA_session -> DAA_scratch

l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

m. set outputData = DAA_session -> DAA_scratch

```
    n. Set DAA_session -> DAA_scratch = NULL
    o. increment DAA_session -> DAA_stage by 1
    p. return TPM_SUCCESS
13. If stage==13
    a. Verify that DAA_session->DAA_stage==13. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_gamma = inputData0
    e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. Verify that inputSize1 == DAA_SIZE_w and return error TPM_DAA_INPUT_DATA1 on mismatch
    g. Set w = inputData1
    h. Set w1 = w^( DAA_issuerSettings -> DAA_generic_q) mod (DAA_generic_gamma)
    i. If w1 != 1 (unity), return error TPM_DAA_WRONG_W
    j. Set DAA_session -> DAA_scratch = w
    k. set outputData = NULL
    l. increment DAA_session -> DAA_stage by 1
    m. return TPM_SUCCESS.
14. If stage==14
    a. Verify that DAA_session ->DAA_stage==14. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings ) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_gamma = inputData0
    e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.
    g. Set E = ((DAA_session -> DAA_scratch)^f) mod (DAA_generic_gamma).
    h. Set outputData = E
    i. increment DAA_session -> DAA_stage by 1
```

j. return TPM_SUCCESS.

15.If stage==15

a. Verify that DAA_session ->DAA_stage==15. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_gamma = inputData0

e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch

f. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them r0

g. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them r1

h. set r = r0 + 2^DAA_power0 * r1 mod (DAA_issuerSettings -> DAA_generic_q).

i. set E1 = ((DAA_session -> DAA_scratch)^r) mod (DAA_generic_gamma).

j. Set DAA_session -> DAA_scratch = NULL

k. Set outputData = E1

l. increment DAA_session -> DAA_stage by 1

m. return TPM_SUCCESS.

16.If stage==16

a. Verify that DAA_session ->DAA_stage==16. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Verify that inputSize0 == sizeOf(TPM_DIGEST) and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set DAA_session -> DAA_digest = inputData0

f. obtain DAA_SIZE_NT bits from the RNG and label them NT

g. Set DAA_session -> DAA_digest to the SHA-1 (DAA_session -> DAA_digest || NT)

h. Set outputData = NT

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS.

17.If stage==17

a. Verify that DAA_session ->DAA_stage==17. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them r0

e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod DAA_issuerSettings -> DAA_generic_q.

f. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)

g. Set s0 = r0 + (DAA_session -> DAA_digest) * f0 in ***Z***

h. set outputData = s0

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS

18. If stage==18

a. Verify that DAA_session ->DAA_stage==18. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them r1

e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod DAA_issuerSettings -> DAA_generic_q.

f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the result f1

g. Set s1 = r1 + (DAA_session -> DAA_digest)* f1 in ***Z***

h. set outputData = s1

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS

19. If stage==19

a. Verify that DAA_session ->DAA_stage==19. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them r2

e. Set s2 = r2 + (DAA_session -> DAA_digest)*(DAA_joinSession -> DAA_join_u0) mod 2^DAA_power1 (Erase all but the lowest DAA_power1 bits of s2)

f. Set DAA_session -> DAA_scratch = s2

g. set outputData = s2

h. increment DAA_session -> DAA_stage by 1

i. return TPM_SUCCESS

20. If stage==20

a. Verify that DAA_session ->DAA_stage==20. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them r2

e. Set s12 = r2 + (DAA_session -> DAA_digest)*(DAA_joinSession -> DAA_join_u0)

f. Shift s12 right by DAA_power1 bit (discard the lowest DAA_power1 bits).

g. Set DAA_session -> DAA_scratch = s12

h. Set outputData = DAA_session -> DAA_digest

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS

21. If stage==21

a. Verify that DAA_session ->DAA_stage==21. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. obtain DAA_SIZE_r3 bits from MGF1("r3", DAA_session -> DAA_contextSeed), and label them r3

e. Set s3 = r3 + (DAA_session -> DAA_digest)*(DAA_joinSession -> DAA_join_u1) + (DAA_session -> DAA_scratch).

f. Set DAA_session -> DAA_scratch = NULL

g. set outputData = s3

h. increment DAA_session -> DAA_stage by 1

i. return TPM_SUCCESS

22. If stage==22

a. Verify that DAA_session ->DAA_stage==22. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Verify inputSize0 == DAA_SIZE_v0 and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set u2 = inputData0

f. Set v0 = u2 + (DAA_joinSession -> DAA_join_u0) mod 2^DAA_power1 (Erase all but the lowest DAA_power1 bits of v0).

g. Set DAA_tpmSpecific -> DAA_digest_v0 = SHA-1(v0)

h. Set v10 = u2 + (DAA_joinSession -> DAA_join_u0) in $\boldsymbol{Z}$

i. Shift v10 right by DAA_power1 bits (erase the lowest DAA_power1 bits).

j. Set DAA_session ->DAA_scratch = v10

k. Set outputData

i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V0 and encrypt the v0 parameters

ii. set outputData to the encrypted TPM_DAA_BLOB

l. increment DAA_session -> DAA_stage by 1

m. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession)

n. return TPM_SUCCESS

23. If stage==23

a. Verify that DAA_session ->DAA_stage==23. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Verify inputSize0 == DAA_SIZE_v1 and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set u3 = inputData0

f. Set v1 = u3 + DAA_joinSession -> DAA_join_u1 + DAA_session ->DAA_scratch

g. Set DAA_tpmSpecific -> DAA_digest_v1 = SHA-1(v1)

```
    h. Set outputData
        i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V1 and encrypt the v1 parameters
        ii. set outputData to the encrypted TPM_DAA_BLOB
    i. Set DAA_session ->DAA_scratch = NULL
    j. increment DAA_session -> DAA_stage by 1
    k. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession)
    l. return TPM_SUCCESS
24. If stage==24
    a. Verify that DAA_session ->DAA_stage==24. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. set outputData = enc(DAA_tpmSpecific)
    e. return TPM_SUCCESS
25. If stage > 24, return error: TPM_DAA_STAGE
```

26.2 TPM_DAA_Sign

TPM protected capability; user must provide authorizations from the TPM Owner.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	Ordinal	Command ordinal: TPM_ORD_DAA_Sign
4	4			TPM_HANDLE	handle	Handle to the sign session
5	1	2S	1	BYTE	stage	Stage of the sign process
6	4	3S	4	UINT32	inputSize0	Size of inputData0 for this stage of DAA_Sign
7	<>	4S	<>	BYTE[]	inputData0	Data to be used by this capability
8	4	5S	4	UINT32	inputSize1	Size of inputData1 for this stage of DAA_Sign
9	<>	6S	<>	BYTE[]	inputData1	Data to be used by this capability
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2 H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
13	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes incl. paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal:TPM_ORD_DAA_Sign
4	4	3S	4	UINT32	outputSize	Size of outputData
5	<>	4S	<>	BYTE[]	outputData	Data produced by this capability
6	20	2 H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3 H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Description

This table summaries the input, output and saved data that is associated with each stage of processing.

Stage	Input Data0	Input Data1	Operation	Output Data	Scratchpad
0	DAA_issuerSettings	NULL	initialise	handle	NULL
1	enc(DAA_tpmSpecific)	NULL	initialise	NULL	NULL
2	DAA_generic_R0	DAA_generic_n	P1=R0^r0 mod n	NULL	P1
3	DAA_generic_R1	DAA_generic_n	P2 = P1*(R1^r1) mod n	NULL	P2
4	DAA_generic_S0	DAA_generic_n	P3 = P2*(S0^r2) mod n	NULL	P3
5	DAA_generic_S1	DAA_generic_n	T = P3*(S1^r4) mod n	T	NULL
6	DAA_generic_gamma	w	w1 = w^q mod gamma	NULL	w
7	DAA_generic_gamma	NULL	E = w^f mod gamma	E	w
8	DAA_generic_gamma	NULL	r = r0 + (2^power0)*r1 mod q, E1 = w^r mod gamma	E1	NULL
9	c1	NULL	c = hash(c1 || NT)	NT	NULL
10	b (selector)	m or handle to AIK	c = hash(c || 1 || m) or c = hash(c || 0 || AIK-modulus)	c	NULL
11	NULL	NULL	s0 = r0 + c*f0	s0	NULL
12	NULL	NULL	s1 = r1 + c*f1	s1	NULL
13	enc(v0)	NULL	s2 = r2 + c*v0 mod 2^power1	s2	NULL
14	enc(v0)	NULL	s12 = r2 + c*v0 >> power1	NULL	s12
15	enc(v1)	NULL	s3 = r4 + c*v1 + s12	s3	NULL

When a TPM receives an Owner authorized command to input enc(DAA_tpmSpecific) or enc(v0) or enc(v1), the TPM MUST verify that the TPM created the data and that neither the data nor the TPM's EK has been changed since the data was created. Loading one of these wrapped blobs does not require authorization, since correct blobs were created by the TPM under Owner authorization, and unwrapped blobs cannot be used without Owner authorisation. The TPM MUST NOT restrict the number of times that the contents of enc(DAA_tpmSpecific) or enc(v0) or enc(v1) can be used by the same combination of TPM and Owner that created them..

Actions

A Trusted Platform Module that receives a valid TPM_DAA_Sign command SHALL:

26. Use ownerAuth to verify that the Owner authorized all TPM_DAA_Sign input parameters.
27. Any error results in the TPM invalidating all resources associated with the command
28. Constant values of 0 or 1 are 1 byte integers, stages affected are
    a. 7(f), 11(e), 12(e)

29. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are

    a. 2(h), 3(h), 4(h), 5(h), 12(d), 13(f), 14(f), 15(f)

Stages

0. If stage==0

    a. Determine that sufficient resources are available to perform a TPM_DAA_Sign.

        i. The TPM MUST support sufficient resources to perform one (1) TPM_DAA_Join/ TPM_DAA_Sign. The TPM MAY support addition TPM_DAA_Join/ TPM_DAA_Sign sessions.

        ii. The TPM may share internal resources between the DAA operations and other variable resource requirements:

            iii. If there are insufficient resources within the stored key pool (and one or more keys need to be removed to permit the DAA operation to execute) return TPM_NOSPACE

            iv. If there are insufficient resources within the stored session pool (and one or more authorization or transport sessions need to be removed to permit the DAA operation to execute), return TPM_RESOURCES.

    b. Set DAA_issuerSettings = inputData0

    c. Verify that all fields in DAA_issuerSettings are present and return error TPM_DAA_INPUT_DATA0 if not.

    d. set all fields in DAA_session = NULL

    e. Assign new handle for session

    f. Set outputData to new handle

    g. set DAA_session -> DAA_stage = 1

    h. return TPM_SUCCESS

1. If stage==1

    a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle on mismatch

    b. Set DAA_tpmSpecific = unwrap(inputData0)

    c. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

    d. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific)

    e. obtain random data from the RNG and store it as DAA_session -> DAA_contextSeed

    f. set outputData = NULL

    g. set DAA_session -> DAA_stage =2

    h. return TPM_SUCCESS

2. If stage==2

a. Verify that DAA_session ->DAA_stage==2. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_R0 = inputData0

e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch

h. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them Y

i. Set X = DAA_generic_R0

j. Set n = DAA_generic_n

k. Set DAA_session -> DAA_scratch = (X^Y) mod n

l. set outputData = NULL

m. increment DAA_session -> DAA_stage by 1

n. return TPM_SUCCESS

3. If stage==3

a. Verify that DAA_session ->DAA_stage==3. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_R1 = inputData0

e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch

h. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them Y

i. Set X = DAA_generic_R1

j. Set n = DAA_generic_n

k. Set Z = DAA_session -> DAA_scratch

```
    l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
    m. set outputData = NULL
    n. increment DAA_session -> DAA_stage by 1
    o. return TPM_SUCCESS
4. If stage==4
    a. Verify that DAA_session ->DAA_stage==4. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_S0 = inputData0
    e. Verify that SHA-1(DAA_generic_S0) == DAA_issuerSettings -> DAA_digest_S0 and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. Set DAA_generic_n = inputData1
    g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch
    h. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them Y
    i. Set X = DAA_generic_S0
    j. Set n = DAA_generic_n
    k. Set Z = DAA_session -> DAA_scratch
    l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
    m. set outputData = NULL
    n. increment DAA_session -> DAA_stage by 1
    o. return TPM_SUCCESS
5. If stage==5
    a. Verify that DAA_session ->DAA_stage==5. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_S1 = inputData0
    e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. Set DAA_generic_n = inputData1
    g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch
    h. obtain DAA_SIZE_r4 bits from MGF1("r4", DAA_session -> DAA_contextSeed), and label them Y
    i. Set X = DAA_generic_S1
    j. Set n = DAA_generic_n
    k. Set Z = DAA_session -> DAA_scratch
    l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
    m. set outputData = DAA_session -> DAA_scratch
    n. set DAA_session -> DAA_scratch = NULL
    o. increment DAA_session -> DAA_stage by 1
    p. return TPM_SUCCESS
6. If stage==6
    a. Verify that DAA_session ->DAA_stage==6. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_gammma = inputData0
    e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. Verify that inputSize1 == DAA_SIZE_w and return error TPM_DAA_INPUT_DATA1 on mismatch
    g. Set w = inputData1
    h. Set w1 = w^( DAA_issuerSettings -> DAA_generic_q) mod (DAA_generic_gamma)
    i. If w1 != 1 (unity), return error TPM_DAA_WRONG_W
    j. Set DAA_session -> DAA_scratch = w
    k. set outputData = NULL
    l. increment DAA_session -> DAA_stage by 1
    m. return TPM_SUCCESS.
7. If stage==7
    a. Verify that DAA_session ->DAA_stage==7. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
```

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_gamma = inputData0

e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod DAA_issuerSettings -> DAA_generic_q.

g. Set E = ((DAA_session -> DAA_scratch)^f) mod (DAA_generic_gamma).

h. Set outputData = E

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS.

8. If stage==8

   a. Verify that DAA_session ->DAA_stage==8. Return TPM_DAA_STAGE and flush handle on mismatch

   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

   d. Set DAA_generic_gamma = inputData0

   e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch

   f. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them r0

   g. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them r1

   h. set r = r0 + 2^DAA_power0 * r1 mod (DAA_issuerSettings -> DAA_generic_q).

   i. Set E1 = ((DAA_session -> DAA_scratch)^r) mod (DAA_generic_gamma)

   j. Set DAA_session -> DAA_scratch = NULL

   k. Set outputData = E1

   l. increment DAA_session -> DAA_stage by 1

   m. return TPM_SUCCESS.

9. If stage==9

   a. Verify that DAA_session ->DAA_stage==9. Return TPM_DAA_STAGE and flush handle on mismatch

   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

   d. Verify that inputSize0 == sizeOf(TPM_DIGEST) and return error TPM_DAA_INPUT_DATA0 on mismatch

   e. Set DAA_session -> DAA_digest = inputData0

   f. obtain DAA_SIZE_NT bits from the RNG and label them NT

   g. Set DAA_session -> DAA_digest to the SHA-1 (DAA_session -> DAA_digest || NT)

   h. Set outputData = NT

   i. increment DAA_session -> DAA_stage by 1

   j. return TPM_SUCCESS.

10. If stage==10

   a. Verify that DAA_session ->DAA_stage==10. Return TPM_DAA_STAGE and flush handle on mismatch

   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

   d. Set selector = inputData0, verify that selector == 0 or 1, and return error TPM_DAA_INPUT_DATA0 on mismatch

   e. If selector == 1, verify that inputSize1 == sizeOf(TPM_DIGEST), and

   f. Set DAA_session -> DAA_digest to SHA-1 (DAA_session -> DAA_digest || 1 || inputData1)

   g. If selector == 0, verify that inputData1 is a handle to a TPM identity key (AIK), and

   h. Set DAA_session -> DAA_digest to SHA-1 (DAA_session -> DAA_digest || 0 || n2) where n2 is the modulus of the AIK

   i. Set outputData = DAA_session -> DAA_digest

   j. increment DAA_session -> DAA_stage by 1

   k. return TPM_SUCCESS.

11. If stage==11

   a. Verify that DAA_session ->DAA_stage==11. Return TPM_DAA_STAGE and flush handle on mismatch

   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

   d. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them r0

e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod DAA_issuerSettings -> DAA_generic_q.

f. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)

g. Set s0 = r0 + (DAA_session -> DAA_digest)*(f0)

h. set outputData = s0

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS

12. If stage==12

a. Verify that DAA_session ->DAA_stage==12. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them r1

e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1) mod DAA_issuerSettings -> DAA_generic_q.

f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the result f1

g. Set s1 = r1 + (DAA_session -> DAA_digest)*(f1)

h. set outputData = s1

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS

13. If stage==13

a. Verify that DAA_session ->DAA_stage==13. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_private_v0= unwrap(inputData0)

e. Verify that SHA-1(DAA_private_v0) == DAA_tpmSpecific -> DAA_digest_v0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them r2

g. Set s2 = r2 + (DAA_session -> DAA_digest)*(DAA_private_v0) mod 2^DAA_power1 (erase all but the lowest DAA_power1 bits of s2)

h. Set DAA_session -> DAA_scratch = s2

i. set outputData = s2

j. increment DAA_session -> DAA_stage by 1

k. return TPM_SUCCESS

14. If stage==14

a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_private_v0= unwrap(inputData0)

e. Verify that SHA-1(DAA_private_v0) == DAA_tpmSpecific -> DAA_digest_v0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them r2

g. Set s12 = r2 + (DAA_session -> DAA_digest)*(DAA_private_v0).

h. Shift s12 right by DAA_power1 bits (erase the lowest DAA_power1 bits).

i. Set DAA_session -> DAA_scratch = s12

j. set outputData = NULL

k. increment DAA_session -> DAA_stage by 1

l. return TPM_SUCCESS

15. If stage==15

a. Verify that DAA_session ->DAA_stage==15. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_private_v1 = unwrap(inputData0)

e. Verify that SHA-1(DAA_private_v1) == DAA_tpmSpecific -> DAA_digest_v1 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. obtain DAA_SIZE_r4 bits from MGF1("r4", DAA_session -> DAA_contextSeed), and label them r4

g. Set s3 = r4 + (DAA_session -> DAA_digest)*(DAA_private_v1) + (DAA_session -> DAA_scratch).

h. Set DAA_session -> DAA_scratch = NULL

i. set outputData = s3

j. increment DAA_session -> DAA_stage by 1

k. return TPM_SUCCESS

16. If stage > 15, return error: TPM_DAA_STAGE

27. Deprecated commands

Start of informative comment:

This section covers the commands that were in version 1.1 but now have new functionality in other functions. The deprecated commands are still available in 1.2 but all new software should use the new functionality.

There is no requirement that the deprecated commands work with new structures.

End of informative comment.

1. Commands deprecated in version 1.2 MUST work with version 1.1 structures
2. Commands deprecated in version 1.2 MAY work with version 1.2 structures

27.1 Key commands

Start of informative comment:

The key commands are deprecated as the new way to handle keys is to use the standard context commands. So TPM_EvictKey is now handled by TPM_FlushSpecific, TPM_Terminate_Handle by TPM_FlushSpecific.

End of informative comment.

27.1.1 TPM_EvictKey

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_EvictKey
4	4			TPM_KEY_HANDLE	evictHandle	The handle of the key to be evicted.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_EvictKey

Actions

The TPM will invalidate the key stored in the specified handle and return the space to the available internal pool for subsequent query by TPM_GetCapability and usage by TPM_LoadKey. If the specified key handle does not correspond to a valid key, an error will be returned.

New 1.2 functionality

The command must check the status of the ownerEvict flag for the key and if the flag is TRUE return TPM_KEY_CONTROL_OWNER

27.1.2 TPM_Terminate_Handle

Start of informative comment:

This allows the TPM manager to clear out information in a session handle.

The TPM may maintain the authorization session even though a key attached to it has been unloaded or the authorization session itself has been unloaded in some way. When a command is executed that requires this session, it is the responsibility of the external software to load both the entity and the authorization session information prior to command execution.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Terminate_Handle.
4	4			TPM_AUTHHANDLE	handle	The handle to terminate

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_Terminate_Handle.

Descriptions

The TPM SHALL terminate the session and destroy all data associated with the session indicated.

Actions

A TPM SHALL unilaterally perform the actions of TPM_Terminate_Handle upon detection of the following events:

1. Completion of a received command whose authorization “continueUse” flag is FALSE.
2. Completion of a received command when a shared secret derived from the authorization session was exclusive-or’ed with data (to provide confidentiality for that data). This occurs during execution of a TPM_ChangeAuth command, for example.
3. When the associated entity is destroyed (in the case of TPM Owner or SRK, for example)
4. Upon execution of TPM_Init

5. When the command returns an error. This is due to the fact that when returning an error the TPM does not send back nonceEven. There is no way to maintain the rolling nonces, hence the TPM MUST terminate the authorization session.
6. Failure of an authorization check belonging to that authorization session.

27.2 Context management

Start of informative comment:

The 1.1 context commands were written for specific resource types. The 1.2 commands are generic for all resource types. So the Savexxx commands are replaced by TPM_SaveContext and the LoadXXX commands by TPM_LoadContext.

End of informative comment.

27.2.1 TPM_SaveKeyContext

Start of informative comment:

TPM_SaveKeyContext saves a loaded key outside the TPM. After creation of the key context blob the TPM automatically releases the internal memory used by that key. The format of the key context blob is specific to a TPM.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveKeyContext
4	4			TPM_KEY_HANDLE	keyHandle	The key which will be kept outside the TPM

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveKeyContext
4	4	3S	4	UINT32	keyContextSize	The actual size of the outgoing key context blob. If the command fails the value will be 0
5	<>	4S	<>	BYTE[]	keyContextBlob	The key context blob.

Description

1. This command allows saving a loaded key outside the TPM. After creation of the keyContextBlob, the TPM automatically releases the internal memory used by that key. The format of the key context blob is specific to a TPM.

2. A TPM protected capability belonging to the TPM that created a key context blob MUST be the only entity that can interpret the contents of that blob. If a cryptographic technique is used for this purpose, the level of security provided by that technique SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys)

used in such a cryptographic technique MUST be generated using the TPM's random number generator. Any symmetric key MUST be used within the power-on session during which it was created, only.

3. A key context blob SHALL enable verification of the integrity of the contents of the blob by a TPM protected capability.
4. A key context blob SHALL enable verification of the session validity of the contents of the blob by a TPM protected capability. The method SHALL ensure that all key context blobs are rendered invalid if power to the TPM is interrupted.

27.2.2 TPM_LoadKeyContext

Start of informative comment:

TPM_LoadKeyContext loads a key context blob into the TPM previously retrieved by a TPM_SaveKeyContext call. After successful completion the handle returned by this command can be used to access the key.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_LoadKeyContext
4	4	2S	4	UINT32	keyContextSize	The size of the following key context blob.
5	<>	3S	<>	BYTE[]	keyContextBlob	The key context blob.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_LoadKeyContext
4	4			TPM_KEY_HANDLE	keyHandle	The handle assigned to the key after it has been successfully loaded.

Description

1. This command allows loading a key context blob into the TPM previously retrieved by a TPM_SaveKeyContext call. After successful completion the handle returned by this command can be used to access the key.

2. The contents of a key context blob SHALL be discarded unless the contents have passed an integrity test. This test SHALL (statistically) prove that the contents of the blob are the same as when the blob was created.

3. The contents of a key context blob SHALL be discarded unless the contents have passed a session validity test. This test SHALL (statistically) prove that the blob was created by this TPM during this power-on session.

27.2.3 TPM_SaveAuthContext

Start of informative comment:

TPM_SaveAuthContext saves a loaded authorization session outside the TPM. After creation of the authorization context blob, the TPM automatically releases the internal memory used by that session. The format of the authorization context blob is specific to a TPM.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveAuthContext
4	4			TPM_AUTHHANDLE	authHandle	Authorization session which will be kept outside the TPM

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveAuthContext
4	4	3S	4	UINT32	authContextSize	The actual size of the outgoing authorization context blob. If the command fails the value will be 0.
5	<>	4S	4	BYTE[]	authContextBlob	The authorization context blob.

Description

This command allows saving a loaded authorization session outside the TPM. After creation of the authContextBlob, the TPM automatically releases the internal memory used by that session. The format of the authorization context blob is specific to a TPM.

A TPM protected capability belonging to the TPM that created an authorization context blob MUST be the only entity that can interpret the contents of that blob. If a cryptographic technique is used for this purpose, the level of security provided by that technique SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys) used in such a cryptographic technique MUST be generated using the TPM's random number generator. Any symmetric key MUST be used within the power-on session during which it was created, only.

An authorization context blob SHALL enable verification of the integrity of the contents of the blob by a TPM protected capability.

An authorization context blob SHALL enable verification of the session validity of the contents of the blob by a TPM protected capability. The method SHALL ensure that all authorization context blobs are rendered invalid if power to the TPM is interrupted.

27.2.4 TPM_LoadAuthContext

Start of informative comment:

TPM_LoadAuthContext loads an authorization context blob into the TPM previously retrieved by a TPM_SaveAuthContext call. After successful completion the handle returned by this command can be used to access the authorization session.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadAuthContext
4	4	2S	4	UINT32	authContextSize	The size of the following authorization context blob.
5	<>	3S	<>	BYTE[]	authContextBlob	The authorization context blob.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadAuthContext
4	4			TPM_KEY_HANDLE	authHandle	The handle assigned to the authorization session after it has been successfully loaded.

Description

This command allows loading an authorization context blob into the TPM previously retrieved by a TPM_SaveAuthContext call. After successful completion the handle returned by this command can be used to access the authorization session.

The contents of an authorization context blob SHALL be discarded unless the contents have passed an integrity test. This test SHALL (statistically) prove that the contents of the blob are the same as when the blob was created.

The contents of an authorization context blob SHALL be discarded unless the contents have passed a session validity test. This test SHALL (statistically) prove that the blob was created by this TPM during this power-on session.

27.3 DIR commands

Start of informative comment:

The DIR commands are replaced by the NV storage commands.

The DIR [0] in 1.1 is now TPM_PERMANENT_DATA -> authDIR[0] and is always available for the TPM to use. It is accessed by DIR commands using dirIndex 0 and by NV commands using nvIndex TPM_NV_INDEX_DIR.

If the TPM vendor supports additional DIR registers, the TPM vendor may return errors or provide vendor specific mappings for those DIR registers to NV storage locations.

End of informative comment.

1. A dirIndex value of 0 MUST corresponds to an NV storage nvIndex value TPM_NV_INDEX_DIR.
2. The TPM vendor MAY return errors or MAY provide vendor specific mappings for DIR dirIndex values greater than 0 to NV storage locations.

27.3.1 TPM_DirWriteAuth

Start of informative comment:

The TPM_DirWriteAuth operation provides write access to the Data Integrity Registers. DIRs are non-volatile memory registers held in a TPM-shielded location. Owner authentication is required to authorize this action.

Access is also provided through the NV commands with nvIndex TPM_NV_INDEX_DIR. Owner authorization is not required when nvLocked is FALSE.

Version 1.2 requires only one DIR. If the DIR named does not exist, the TPM_DirWriteAuth operation returns TPM_BADINDEX.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM _ORD_DirWriteAuth.
4	4	2S	4	TPM_DIRINDEX	dirIndex	Index of the DIR
5	20	3S	20	TPM_DIRVALUE	newContents	New value to be stored in named DIR
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for command.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DirWriteAuth
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Actions

1. Validate that authHandle contains a TPM Owner AuthData to execute the TPM_DirWriteAuth command
2. Validate that dirIndex points to a valid DIR on this TPM
3. Write newContents into the DIR pointed to by dirIndex

27.3.2 TPM_DirRead

Start of informative comment:

The TPM_DirRead operation provides read access to the DIRs. No authentication is required to perform this action because typically no cryptographically useful AuthData is available early in boot. TSS implementers may choose to provide other means of authorizing this action. Version 1.2 requires only one DIR. If the DIR named does not exist, the TPM_DirRead operation returns TPM_BADINDEX.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_DirRead.
4	4	2S	4	TPM_DIRINDEX	dirIndex	Index of the DIR to be read

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_DirRead.
4	20	3S	20	TPM_DIRVALUE	dirContents	The current contents of the named DIR

Actions

1. Validate that dirIndex points to a valid DIR on this TPM
2. Return the contents of the DIR in dirContents

27.4 Change Auth

Start of informative comment:

The change auth commands can be duplicated by creating a transport session with confidentiality and issuing the changeAuth command.

End of informative comment.

27.4.1 TPM_ChangeAuthAsymStart

Start of informative comment:

The TPM_ChangeAuthAsymStart starts the process of changing AuthData for an entity. It sets up an OIAP session that must be retained for use by its twin TPM_ChangeAuthAsymFinish command.

TPM_ChangeAuthAsymStart creates a temporary asymmetric public key "tempkey" to provide confidentiality for new AuthData to be sent to the TPM. TPM_ChangeAuthAsymStart certifies that tempkey was generated by a genuine TPM, by generating a certifyInfo structure that is signed by a TPM identity. The owner of that TPM identity must cooperate to produce this command, because TPM_ChangeAuthAsymStart requires authorization to use that identity.

It is envisaged that tempkey and certifyInfo are given to the owner of the entity whose authorization is to be changed. That owner uses certifyInfo and a TPM_IDENTITY_CREDENTIAL to verify that tempkey was generated by a genuine TPM. This is done by verifying the TPM_IDENTITY_CREDENTIAL using the public key of a CA, verifying the signature on the certifyInfo structure with the public key of the identity in TPM_IDENTITY_CREDENTIAL, and verifying tempkey by comparing its digest with the value inside certifyInfo. The owner uses tempkey to encrypt the desired new AuthData and inserts that encrypted data in a TPM_ChangeAuthAsymFinish command, in the knowledge that only a TPM with a specific identity can interpret the new AuthData .

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymStart.
4	4			TPM_KEY_HANDLE	idHandle	The keyHandle identifier of a loaded identity ID key
5	20	2s	20	TPM_NONCE	antiReplay	The nonce to be inserted into the certifyInfo structure
6	<>	3S	<>	TPM_KEY_PARMS	tempKey	Structure contains all parameters of ephemeral key.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for idHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	idAuth	Authorization. HMAC key: idKey.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymStart
7	95	3S	95	TPM_CERTIFY_INFO	certifyInfo	The certifyInfo structure that is to be signed.
8	4	4S	4	UINT32	sigSize	The used size of the output area for the signature
9	<>	5S	<>	BYTE[ ]	sig	The signature of the certifyInfo parameter.
10	4	6s	4	TPM_KEY_HANDLE	ephHandle	The keyHandle identifier to be used by ChangeAuthAsymFinish for the ephemeral key
11	<>	7S	<>	TPM_KEY	tempKey	Structure containing all parameters and public part of ephemeral key. TPM_KEY.encSize is set to 0.
12	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
13	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
14	20.			TPM_AUTHDATA	resAuth	Authorization. HMAC key: idKey.usageAuth.

Actions

1. The TPM SHALL verify the AuthData to use the TPM identity key held in idHandle. The TPM MUST verify that the key is a TPM identity key.
2. The TPM SHALL validate the algorithm parameters for the key to create from the tempKey parameter.
3. Recommended key type is RSA
4. Minimum RSA key size MUST is 512 bits, recommended RSA key size is 1024
5. For other key types the minimum key size strength MUST be comparable to RSA 512
6. If the TPM is not designed to create a key of the requested type, return the error code TPM_BAD_KEY_PROPERTY
7. The TPM SHALL create a new key (k1) in accordance with the algorithm parameter. The newly created key is pointed to by ephHandle.
8. The TPM SHALL fill in all fields in tempKey using k1 for the information. The TPM_KEY -> encSize MUST be 0.
9. The TPM SHALL fill in certifyInfo using k1 for the information. The certifyInfo -> data field is supplied by the antiReplay.
10. The TPM then signs the certifyInfo parameter using the key pointed to by idHandle. The resulting signed blob is returned in sig parameter

674 **Field Descriptions for certifyInfo parameter**

Type	Name	Description
TPM_VERSION	Version	TPM version structure; Part 2 TPM_VERSION
keyFlags	Redirection	This SHALL be set to FALSE
	Migratable	This SHALL be set to FALSE
	Volatile	This SHALL be set to TRUE
TPM_AUTH_DATA_USAGE	authDataUsage	This SHALL be set to TPM_AUTH_NEVER
TPM_KEY_USAGE	KeyUsage	This SHALL be set to TPM_KEY_AUTHCHANGE
UINT32	PCRInfoSize	This SHALL be set to 0
TPM_DIGEST	pubDigest	This SHALL be the hash of the public key being certified.
TPM_NONCE	Data	This SHALL be set to antiReplay
TPM_KEY_PARMS	info	This specifies the type of key and its parameters.
BOOL	parentPCRStatus	This SHALL be set to FALSE.

675

27.4.2 TPM_ChangeAuthAsymFinish

Start of informative comment:

The TPM_ChangeAuth command allows the owner of an entity to change the AuthData for the entity.

The command requires the cooperation of the owner of the parent of the entity, since AuthData must be provided to use that parent entity. The command requires knowledge of the existing AuthData information and passes the new AuthData information. The newAuthLink parameter proves knowledge of existing AuthData information and new AuthData information. The new AuthData information "encNewAuth" is encrypted using the "tempKey" variable obtained via TPM_ChangeAuthAsymStart.

A parent therefore retains control over a change in the AuthData of a child, but is prevented from knowing the new AuthData for that child.

The changeProof parameter provides a proof that the new AuthData value was properly inserted into the entity. The inclusion of a nonce from the TPM provides an entropy source in the case where the AuthData value may be in itself be a low entropy value (hash of a password etc).

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymFinish
4	4			TPM_KEY_HANDLE	parentHandle	The keyHandle of the parent key for the input data
5	4			TPM_KEY_HANDLE	ephHandle	The keyHandle identifier for the ephemeral key
6	2	3S	2	TPM_ENTITY_TYPE	entityType	The type of entity to be modified
7	20	4s	20	TPM_HMAC	newAuthLink	HMAC calculation that links the old and new AuthData values together
8	4	5S	4	UINT32	newAuthSize	Size of encNewAuth
9	<>	6S	<>	BYTE[ ]	encNewAuth	New AuthData encrypted with ephemeral key.
10	4	7S	4	UINT32	encDataSize	The size of the inData parameter
11	<>	8S	<>	BYTE[ ]	encData	The encrypted entity that is to be modified.
12	4			TPM_AUTHHANDLE	authHandle	Authorization for parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
13	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
14	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
15	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey .usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymFinish
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[ ]	outData	The modified, encrypted entity.
6	20	5s	20	TPM_NONCE	saltNonce	A nonce value from the TPM RNG to add entropy to the changeProof value
7	<>	6S	<>	TPM_DIGEST	changeProof	Proof that AuthData has changed.
8	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

Description

If the parentHandle points to the SRK then the HMAC key MUST be built using the TPM Owner authentication.

Actions

1. The TPM SHALL validate that the authHandle parameter authorizes use of the key in parentHandle.
2. The encData field MUST be the encData field from TPM_STORED_DATA or TPM_KEY.
3. The TPM SHALL create e1 by decrypting the entity held in the encData parameter.
4. The TPM SHALL create a1 by decrypting encNewAuth using the ephHandle -> TPM_KEY_AUTHCHANGE private key. a1 is a structure of type TPM_CHANGEAUTH_VALIDATE.
5. The TPM SHALL create b1 by performing the following HMAC calculation: b1 = HMAC (a1 -> newAuthSecret). The secret for this calculation is encData -> currentAuth. This means that b1 is a value built from the current AuthData value (encData -> currentAuth) and the new AuthData value (a1 -> newAuthSecret).
6. The TPM SHALL compare b1 with newAuthLink. The TPM SHALL indicate a failure if the values do not match.
7. The TPM SHALL replace e1 -> authData with a1 -> newAuthSecret
8. The TPM SHALL encrypt e1 using the appropriate functions for the entity type. The key to encrypt with is parentHandle.
9. The TPM SHALL create saltNonce by taking the next 20 bytes from the TPM RNG.

10. The TPM SHALL create changeProof a HMAC of (saltNonce concatenated with a1 -> n1) using a1 -> newAuthSecret as the HMAC secret.

11. The TPM MUST destroy the TPM_KEY_AUTHCHANGE key associated with the authorization session.

27.5 TPM_Reset

Start of informative comment:

TPM_Reset releases all resources associated with existing authorization sessions. This is useful if a TSS driver has lost track of the state in the TPM.

End of informative comment.

Deprecated Command in 1.2

Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Reset.

Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Reset.

Description

This is a deprecated command in V1.2. This command in 1.1 only referenced authorization sessions and is not upgraded to affect any other TPM entity in 1.2

Actions

1. The TPM invalidates all resources allocated to authorization sessions as per version 1.1 extant in the TPM
   a. This includes structures created by TPM_SaveAuthContext and TPM_SaveKeyContext
   b. Structures created by TPM_Contextxxx (the new 1.2 commands) are not affected by this command
2. The TPM does not reset any PCR or DIR values.
3. The TPM does not reset any flags in the TPM_STCLEAR_FLAGS structure.
4. The TPM does not reset or invalidate any keys

27.6 TPM_OwnerReadPubek

Start of informative comment:

Return the endorsement key public portion. This is authorized by the TPM Owner.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerReadPubek
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerReadPubek
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Description

This command returns the PUBEK.

Actions

The TPM_OwnerReadPubek command SHALL

1. Validate the TPM Owner AuthData to execute this command
2. Export the PUBEK

27.7 TPM_DisablePubekRead

Start of informative comment:

The TPM Owner may wish to prevent any entity from reading the PUBEK. This command sets the non-volatile flag so that the TPM_ReadPubek command always returns TPM_DISABLED_CMD.

This command has in essence been deprecated as TPM_TakeOwnership now sets the value to false. The command remains at this time for backward compatibility.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisablePubekRead
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
5	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authorization. HMAC key: ownerAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisablePubekRead
4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

Actions

1. This capability sets the TPM_PERMANENT_FLAGS -> readPubek flag to FALSE.

27.8 TPM_LoadKey

Start of informative comment:

Version 1.2 deprecates TPM_LoadKey due to the HMAC of the new key handle on return. The wrapping makes use of the handle difficult in an environment where the TSS, or other management entity, is changing the TPM handle to a virtual handle.

Software using TPM_LoadKey on a 1.2 TPM can have a collision with the returned handle as the 1.2 TPM uses random values in the lower three bytes of the handle. All new software must use LoadKey2 to allow management software the ability to manage the key handle.

Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM. The TPM_LoadKey function loads the key into the TPM for further use.

The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle. The assumption is that the handle may change due to key management operations. It is the responsibility of upper level software to maintain the mapping between handle and any label used by external software.

This command has the responsibility of enforcing restrictions on the use of keys. For example, when attempting to load a STORAGE key it will be checked for the restrictions on a storage key (2048 size etc.).

The load command must maintain a record of whether any previous key in the key hierarchy was bound to a PCR using parentPCRStatus.

The flag parentPCRStatus enables the possibility of checking that a platform passed through some particular state or states before finishing in the current state. A grandparent key could be linked to state-1, a parent key could linked to state-2, and a child key could be linked to state-3, for example. The use of the child key then indicates that the platform passed through states 1 and 2 and is currently in state 3, in this example. TPM_Startup with stType == TPM_ST_CLEAR indicates that the platform has been reset, so the platform has not passed through the previous states. Hence keys with parentPCRStatus==TRUE must be unloaded if TPM_Startup is issued with stType == TPM_ST_CLEAR.

If a TPM_KEY structure has been decrypted AND the integrity test using "pubDataDigest" has passed AND the key is non-migratory, the key must have been created by the TPM. So there is every reason to believe that the key poses no security threat to the TPM. While there is no known attack from a rogue migratory key, there is a desire to verify that a loaded migratory key is a real key, arising from a general sense of unease about execution of arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt cycle, but this may be expensive. For RSA keys, it is therefore suggested that the consistency test consists of dividing the supposed RSA product by the supposed RSA prime, and checking that there is no remainder.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinat TPM_ORD_LoadKey.
4	4			TPM_KEY_HANDLE	parentHandle	TPM handle of parent key.
5	<>	2S	<>	TPM_KEY	inKey	Incoming key structure, both encrypted private and clear public portions. MAY be TPM_KEY12
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parentHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKey
4	4	3S	4	TPM_KEY_HANDLE	inkeyHandle	Internal TPM handle where decrypted key was loaded.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

Actions

The TPM SHALL perform the following steps:

1. Validate the command and the parameters using parentAuth and parentHandle -> usageAuth
2. If parentHandle -> keyUsage is NOT TPM_KEY_STORAGE return TPM_INVALID_KEYUSAGE
3. If the TPM is not designed to operate on a key of the type specified by inKey, return the error code TPM_BAD_KEY_PROPERTY
4. The TPM MUST handle both TPM_KEY and TPM_KEY12 structures
5. Decrypt the inKey -> privkey to obtain TPM_STORE_ASYMKEY structure using the key in parentHandle

6. Validate the integrity of inKey and decrypted TPM_STORE_ASYMKEY
    - a. Reproduce inKey -> TPM_STORE_ASYMKEY -> pubDataDigest using the fields of inKey, and check that the reproduced value is the same as pubDataDigest
7. Validate the consistency of the key and it's key usage.
    - a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the public and private components of the asymmetric key pair. If inKey -> keyFlags -> migratable is FALSE, the TPM MAY verify consistency of the public and private components of the asymmetric key pair. The consistency of an RSA key pair MAY be verified by dividing the supposed (P*Q) product by a supposed prime and checking that there is no remainder..
    - b. If inKey -> keyUsage is TPM_KEY_IDENTITY, verify that inKey->keyFlags->migratable is FALSE. If it is not, return TPM_INVALID_KEYUSAGE
    - c. If inKey -> keyUsage is TPM_KEY_AUTHCHANGE, return TPM_INVALID_KEYUSAGE
    - d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM_STORE_ASYMKEY -> migrationAuth equals TPM_PERMANENT_DATA -> tpmProof
    - e. Validate the mix of encryption and signature schemes
    - f. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then
        - i. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS
        - ii. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS
        - iii. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS
    - g. If inKey -> keyUsage is TPM_KEY_STORAGE or TPM_KEY_MIGRATE
        - i. algorithmID MUST be TPM_ALG_RSA
        - ii. Key size MUST be 2048
        - iii. sigScheme MUST be TPM_SS_NONE
    - h. If inKey -> keyUsage is TPM_KEY_IDENTITY
        - i. algorithmID MUST be TPM_ALG_RSA
        - ii. Key size MUST be 2048
        - iii. encScheme MUST be TPM_ES_NONE
    - i. If the decrypted inKey -> pcrInfo is NULL,
        - i. The TPM MUST set the internal indicator to indicate that the key is not using any PCR registers.
    - j. Else
        - i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a composite hash whenever the key will be in use
        - ii. The TPM MUST handle both version 1.1 TPM_PCR_INFO and 1.2 TPM_PCR_INFO_LONG structures according to the type of TPM_KEY structure
        - iii. The TPM MUST validate the TPM_PCR_INFO or TPM_PCR_INFO_LONG structures

8. Perform any processing necessary to make TPM_STORE_ASYMKEY key available for operations
9. Load key and key information into internal memory of the TPM. If insufficient memory exists return error TPM_NOSPACE.
10. Assign inKeyHandle according to internal TPM rules.
11. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.
12. If ParentHandle indicates it is using PCR registers then set inKeyHandle -> parentPCRStatus to TRUE.

28. Deleted Commands

Start of informative comment:

These commands are no longer active commands. Their removal is due to security concerns with their use.

End of informative comment.

1. The TPM MUST return TPM_BAD_ORDINAL for any deleted command

28.1 TPM_GetCapabilitySigned

Start of informative comment:

Along with TPM_GetCapabilityOwner this command allowed the possible signature of improper values.

TPM_GetCapabilitySigned is almost the same as TPM_GetCapability. The differences are that the input includes a challenge (a nonce) and the response includes a digital signature to vouch for the source of the answer.

If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM and the caller have AuthData.

If a caller requires proof for a third party, the signing key must be one whose signature is trusted by the third party. A TPM-identity key may be suitable.

End of informative comment.

Deleted Ordinal

TPM_GetCapabilitySigned

28.2 TPM_GetOrdinalAuditStatus

Start of informative comment:

Get the status of the audit flag for the given ordinal.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4			TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetOrdinalAuditStatus
4	4			TPM_COMMAND_CODE	ordinalToQuery	The ordinal whose audit flag is to be queried

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4			TPM_RESULT	returnCode	The return code of the operation.
4	1			BOOL	State	Value of audit flag for ordinalToQuery

Actions

1. The TPM returns the Boolean value for the given ordinal. The value is TRUE if the command is being audited.

28.3 TPM_CertifySelfTest

Start of informative comment:

TPM_CertifySelfTest causes the TPM to perform a full self-test and return an authenticated value if the test passes.

If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM and the caller have AuthData.

If a caller requires proof for a third party, the signing key must be one whose signature is trusted by the third party. A TPM-identity key may be suitable.

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CertifySelfTest
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
5	20	2S	20	TPM_NONCE	antiReplay	Anti Replay nonce to prevent replay of messages
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth

Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CertifySelfTest
4	4	3S	4	UINT32	sigSize	The length of the returned digital signature
5	<>	4S	<>	BYTE[ ]	sig	The resulting digital signature.
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if hande is still active
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

Description

The key in keyHandle MUST have a KEYUSAGE value of type TPM_KEY_SIGNING or TPM_KEY_LEGACY or TPM_KEY_IDENTITY.

Information returned by TPM_CertifySelfTest MUST NOT aid identification of an individual TPM.

Actions

1. The TPM SHALL perform TPM_SelfTestFull. If the test fails the TPM returns the appropriate error code.
2. After successful completion of the self-test the TPM then validates the authorization to use the key pointed to by keyHandle
   - a. If the key pointed to by keyHandle has a signature scheme that is not TPM_SS_RSASSAPKCS1v15_SHA1, the TPM may either return TPM_BAD_SCHEME or may return TPM_SUCCESS and a vendor specific signature.
3. Create t1 the NOT null terminated string of "Test Passed", i.e. 11 bytes.
4. The TPM creates m2 the message to sign by concatenating t1 || AntiReplay || ordinal.
5. The TPM signs the SHA-1 of m2 using the key identified by keyHandle, and returns the signature as sig.